**IN THE UNITED STATES DISTRICT COURT**
**FOR THE EASTERN DISTRICT OF TEXAS**
**MARSHALL DIVISION**

| | | |
|---|---|---|
| **FINANCIAL SYSTEMS TECHNOLOGY (INTELLECTUAL PROPERTY) PTY. LTD.,** | § | |
| **and** | § | **CIVIL ACTION** |
| **FINANCIAL SYSTEMS TECHNOLOGY PTY. LTD.,** | § | 2-04CV-358-TJW |
| **Plaintiffs,** | § | |
| | § | **JURY TRIAL** |
| **vs.** | § | |
| **ORACLE CORPORATION,** | § | |
| **Defendant.** | § | |

## COMPLAINT

Plaintiffs Financial Systems Technology (Intellectual Property) Pty. Ltd. ("FST-IP") and Financial Systems Technology Pty. Ltd. ("FST") (collectively, "Plaintiffs"), file this Complaint for infringement of U.S. Patent Nos. 5,617,567 ("the '567 Patent") and 5,826,259 ("the '259 Patent") (collectively, "the Patents-in-Suit") under 35 U.S.C. § 271.

## PARTIES

1. Plaintiff FST-IP is a company organized and existing under the laws of Australia. FST-IP maintains its principal place of business at 17-33 Milton Parade, Suite 334, Malvern, Victoria, 3184, Australia. FST-IP is the owner of all intellectual property rights referenced herein.

2. Plaintiff FST is a software development company organized and existing under the laws of Australia, and is wholly owned by Plaintiff FST-IP. FST maintains its principal

place of business at 17-33 Milton Parade, Suite 334, Malvern, Victoria, 3184, Australia. FST is the exclusive licensee of all intellectual property rights referenced herein.

3. Upon information and belief, Defendant Oracle Corporation ("Oracle") is, and at all relevant times mentioned herein was, a corporation organized and existing under the laws of the State of Delaware, with its principal place of business at 500 Oracle Parkway, Redwood Shores, CA 94065. Oracle manufactures for sale and/or sells database software to consumers in the United States and, more particularly, in the Eastern District of Texas. Oracle may be served with service of process by serving a copy of the Complaint on its registered agent for service of process: Corporation Service Co., 701 Brazos St., Suite 1050, Austin, TX 78701.

## JURISDICTION AND VENUE

4. This is an action for patent infringement arising under the patent laws of the United States, Title 35, United States Code. This Court has exclusive subject matter jurisdiction over this case for patent infringement under 28 U.S.C. §1338(a).

5. This Court has personal jurisdiction over Oracle. Oracle has conducted and does conduct business within the State of Texas. Oracle, directly or through intermediaries (including distributors, retailers, and others), ships, distributes, offers for sale, sells, and advertises (including making available to the public an interactive web page) its products in the United States, the State of Texas, and the Eastern District of Texas. Oracle has purposefully and voluntarily placed infringing products in the stream of commerce with the expectation that they will be purchased by consumers in the Eastern District of Texas. These infringing products have been and continue to be purchased and used by consumers in the Eastern District of Texas. Oracle has committed the tort of patent infringement within the State of Texas and, more particularly, within the Eastern District of Texas. Oracle maintains a registered agent for service of process within the State of Texas, as identified in Paragraph 3.

6. Venue is proper in the Eastern District of Texas under 28 U.S.C. §§ 1391 and 1400(b).

**COUNT 1: INFRINGEMENT OF U.S. PATENT NO. 5,617,567**

7. Plaintiffs refer to and incorporate herein the allegations of Paragraphs 1-6 above.

8. The '567 Patent, entitled "Data Processing System and Method for Retrieving and Entity Specified in a Search Path Record from a Relational Database," was duly and legally issued by the United States Patent and Trademark Office on April 1, 1997, after full and fair examination. A copy of the '567 Patent is attached as Exhibit A. FST-IP is the assignee of all rights, title, and interest in and to the '567 Patent and possesses all rights of recovery under the '567 Patent. FST is the exclusive licensee under the '567 Patent.

9. Defendant Oracle has been and is now directly infringing, and indirectly infringing by way of inducing infringement and/or contributing to the infringement, on the '567 Patent in this District and elsewhere by making, using, offering for sale, and selling products, including, but not limited to Oracle Database Management System ("DBMS") versions 8i, 9i, and 10g and associated products, covered by at least one claim of the '567 Patent, all to the injury of Plaintiffs.

10. Plaintiffs have at all times complied with 35 U.S.C. § 287.

11. Upon information and belief, Oracle has had and has actual notice of the '567 Patent, has been and is aware of its infringement, and Oracle's infringement has been and continues to be willful.

12. Plaintiffs have been irreparably harmed by Oracle's acts of infringement of the '567 Patent, and will continue to be harmed unless and until Oracle's acts of infringement are enjoined and restrained by order of this Court.

13. As a result of Oracle's acts of infringement, Plaintiffs have suffered and will continue to suffer damages in an amount to be proved at trial.

**COUNT 2: INFRINGEMENT OF U.S. PATENT NO. 5,826,259**

14. Plaintiffs refer to and incorporate herein the allegations of Paragraphs 1-6 above.

15. The '259 Patent, entitled "Easily Expandable Data Processing System and Method," was duly and legally issued by the United States Patent and Trademark Office on October 20, 1998, after full and fair examination. A copy of the '259 Patent is attached as Exhibit B. FST-IP is the assignee of all rights, title, and interest in and to the '259 Patent and possesses all rights of recovery under the '259 Patent. FST is the exclusive licensee under the '259 Patent.

16. Defendant Oracle has been and is now directly infringing, and indirectly infringing by way of inducing infringement and/or contributing to the infringement, on the '259 Patent in this District and elsewhere by making, using, offering for sale, and selling products, including, but not limited to DBMS versions 8, 8i, 9i, and 10g and associated products, covered by at least one claim of the '259 Patent, all to the injury of Plaintiffs.

17. Plaintiffs have at all times complied with 35 U.S.C. § 287.

18. Upon information and belief, Oracle has had and has actual notice of the '259 Patent, has been and is aware of its infringement, and Oracle's infringement has been and continues to be willful.

19. Plaintiffs have been irreparably harmed by Oracle's acts of infringement of the '259 Patent, and will continue to be harmed unless and until Oracle's acts of infringement are enjoined and restrained by order of this Court.

20. As a result of Oracle's acts of infringement, Plaintiffs have suffered and will continue to suffer damages in an amount to be proved at trial.

## PRAYER FOR RELIEF

Plaintiffs pray for the following relief:

A. A judgment that Defendant Oracle has infringed the Patents-in-Suit;

B. A judgment and order requiring Oracle to pay Plaintiffs for all damages under 35 U.S.C. § 284, including supplemental damages for any continuing post-verdict infringement up until entry of the final Judgment, with an accounting, as needed, and treble damages for willful infringement;

C. A judgment and order requiring Oracle to pay to Plaintiffs pre-judgment and post-judgment interest on the damages awarded;

D. A judgment and order finding this to be an exceptional case and requiring Oracle to pay the costs of this action (including all disbursements) and attorneys fees as provided by 35 U.S.C. § 285;

E. A judgment and order that Oracle, its agents, employees, representatives, successors and assigns, and those acting in privity or in concert with them, be preliminarily and permanently enjoined from further infringement of the Patents-in-Suit; and

F. Such other and further relief as the Court deems just and equitable.

## DEMAND FOR JURY TRIAL

Plaintiffs hereby demand that all issues be determined by jury.

DATED: October 12, 2004.

Respectfully submitted,

**McKOOL SMITH, P.C.**

Sam Baxter
Attorney-in-Charge
Texas State Bar No. 01938000
sbaxter@mckoolsmith.com
505 E. Travis, Suite 105
P.O. Box O
Marshall, Texas 75670
Telephone: (903) 927-2111
Telecopier: (903) 927-2622

Mike McKool, Jr.
Texas State Bar No. 13732100
mmckool@mckoolsmith.com
Douglas A. Cawley
Texas State Bar No. 04035500
dcawley@mckoolsmith.com
300 Crescent Court, Suite 1500
Dallas, Texas 75201
Telephone: (214) 978-4000
Telecopier: (214) 978-4044

Of Counsel:
Robert S. Bramson
Bramson & Pressman
100 East Hector Street, Suite 410
Conshohocken, PA 19428

T. Gordon White
Texas State Bar No. 21333000
gwhite@mckoolsmith.com
Steven J. Pollinger
Texas State Bar No. 24011919
spollinger@mckoolsmith.com
300 West Sixth Street, Suite 1700
Austin, Texas 78701
Telephone: (512) 692-8700
Telecopier: (512) 692-8744

**ATTORNEYS FOR PLAINTIFFS FINANCIAL SYSTEMS TECHNOLOGY (INTELLECTUAL PROPERTY) PTY. LTD.**
**AND**
**FINANCIAL SYSTEMS TECHNOLOGY PTY. LTD.**

# EXHIBIT A

US005617567A

# United States Patent [19]

Doktor

[11] Patent Number: 5,617,567

[45] Date of Patent: Apr. 1, 1997

[54] **DATA PROCESSING SYSTEM AND METHOD FOR RETRIEVING AND ENTITY SPECIFIED IN A SEARCH PATH RECORD FROM A RELATIONAL DATABASE**

[75] Inventor: **Karol Doktor**, Wheelers Hill, Australia

[73] Assignee: **Financial System Technology Pty. Ltd.**, South Melbourne, Australia

[21] Appl. No.: **439,013**

[22] Filed: **May 11, 1995**

**Related U.S. Application Data**

[62] Division of Ser. No. 83,361, Jun. 28, 1993, abandoned, which is a continuation of Ser. No. 526,424, May 21, 1990, abandoned.

[51] **Int. Cl.**[6] ..... **G06F 17/30**

[52] **U.S. Cl.** ..... **395/602**

[58] **Field of Search** ..... 395/600, 800, 395/650, 700

[56] **References Cited**

U.S. PATENT DOCUMENTS

| | | | |
|---|---|---|---|
| 3,618,027 | 11/1971 | Feng | 364/900 |
| 3,670,310 | 6/1972 | Bharwani et al. | 395/600 |
| 4,128,891 | 12/1978 | Lin et al. | 364/900 |
| 4,497,039 | 1/1985 | Kitakami et al. | 364/900 |
| 4,498,145 | 2/1985 | Baker et al. | 364/900 |
| 4,575,798 | 3/1986 | Lindstrom et al. | 364/300 |
| 4,631,664 | 12/1986 | Bachman | 364/200 |

(List continued on next page.)

OTHER PUBLICATIONS

Korth and Silberschatz, *Database System Concepts*, McGraw-Hill Book Company (New York, 1986), pp. 45–105, pp. 301–323.

"Extended Disjunctive Normal Form for Efficient Processing of Recursive Logic Queries", *IBM Technical Disclosure Bulletin*, vol. 30, No. 1, Jun. 1987, pp. 360–366.

Kifer et al., "SYGRAF: Implementing Logic Programs in a Database Style", *IEEE Transactions on Software Engineering*, vol. 14, No. 7, Jul., 1988, pp. 922–935.

Yu et al., "Automatic Knowledge Acquisition and Maintenance for Semantic Query Optimization", *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, No. 3, Sep. 1989, pp. 362–375.

El-Sharkawi et al., "The Architecture and Implementation of ENLI: An Example-Based Natural Language-Assisted Interface", *PARBASE-90 International Conference on Databases, Parallel Architectures and Their Applications*, 7–9 Mar. 1990, pp. 430–432.

Wilschut et al., "Pipelining in Query Execution", *PARBASE-90 International Conference on Databases, Parallel Architectures and Their Applications*, 7–9 Mar. 1990, p. 562.

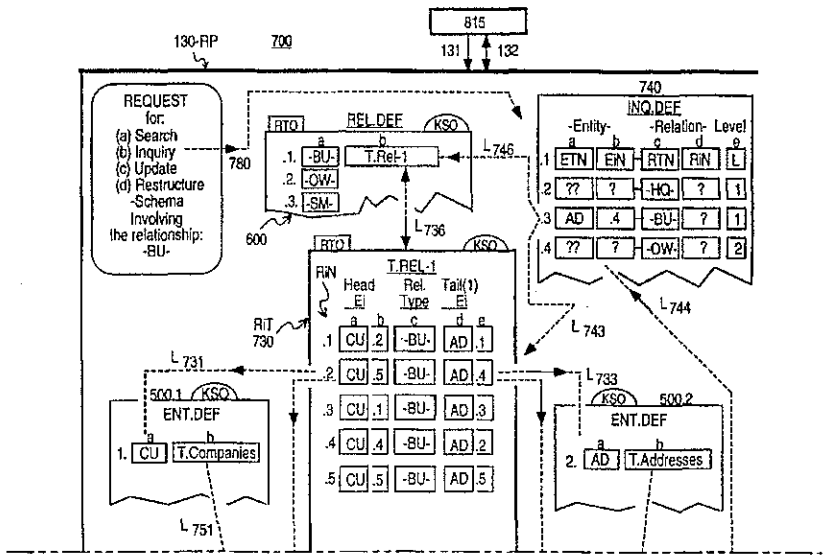
*Primary Examiner*—Thomas G. Black
*Assistant Examiner*—Paul R. Lintz
*Attorney, Agent, or Firm*—Skjerven, Morrill, MacPherson, Franklin & Friel LLP

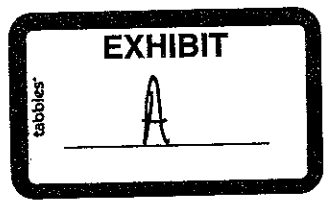
[57] **ABSTRACT**

A relationships processing computing system provides for the recording and extraction of data objects (entities) and for development data representing a queried relationship between data objects (entities). The set of entities and relationships may be expanded at any time during the life of the system without reprogramming or compiling computer code and without disrupting concurrent use of the system. Complex inquiries, normally requiring multiple nested queries, may be performed without code level programming.

**16 Claims, 19 Drawing Sheets**

Microfiche Appendix Included
(20 Microfiche, 1162 Pages)



U.S. PATENT DOCUMENTS


| | | | |
|---|---|---|---|
| 4,670,848 | 6/1987 | Schramm | 364/513 |
| 4,791,561 | 12/1988 | Huber | 364/300 |
| 4,807,122 | 2/1989 | Baba | 364/200 |
| 4,829,427 | 5/1989 | Green | 364/300 |
| 4,893,232 | 1/1990 | Shimoaka et al. | 395/600 |
| 4,901,229 | 2/1990 | Tashiro et al. | 364/200 |
| 4,918,593 | 4/1990 | Huber | 364/200 |
| 4,930,071 | 5/1990 | Tou et al. | 364/300 |
| 4,930,072 | 5/1990 | Agrawal et al. | 395/600 |
| 4,933,848 | 6/1990 | Haderle et al. | 364/300 |
| 4,947,320 | 8/1990 | Crus et al. | 364/200 |
| 4,967,341 | 10/1990 | Yamamoto et al. | 364/200 |
| 5,133,068 | 7/1992 | Crus et al. | 395/600 |
| 5,168,565 | 12/1992 | Morith | 395/600 |
| 5,226,158 | 7/1993 | Horn et al. | 395/600 |
| 5,239,663 | 8/1993 | Faudemay et al. | 395/800 |
| 5,386,557 | 1/1995 | Boykin et al. | 395/600 |
| 5,386,559 | 1/1995 | Eisenberg et al. | 395/600 |
| 5,408,657 | 4/1995 | Bigelow et al. | 395/600 |
| 5,488,722 | 1/1996 | Potok | 395/600 |

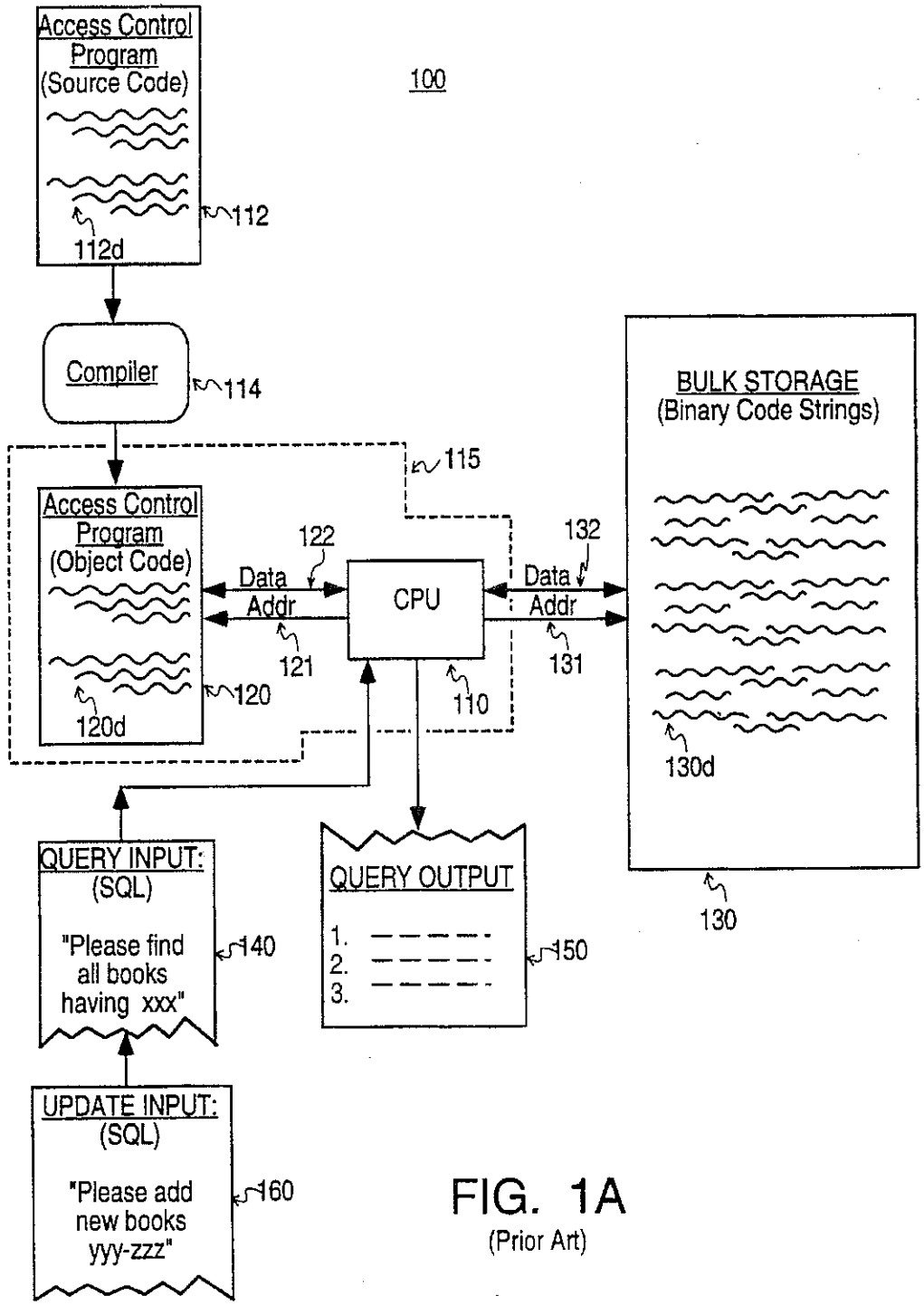



FIG. 1A
(Prior Art)

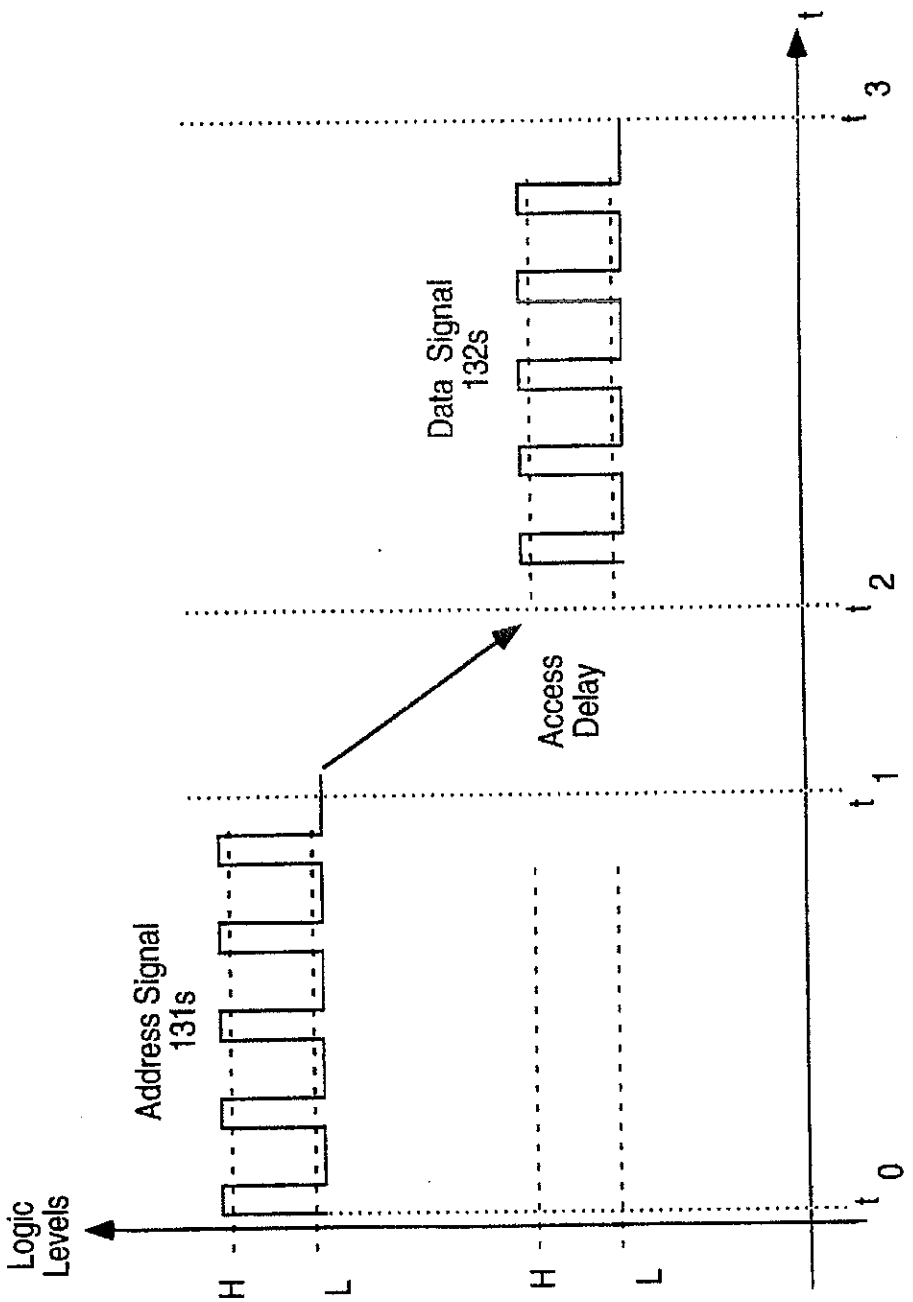
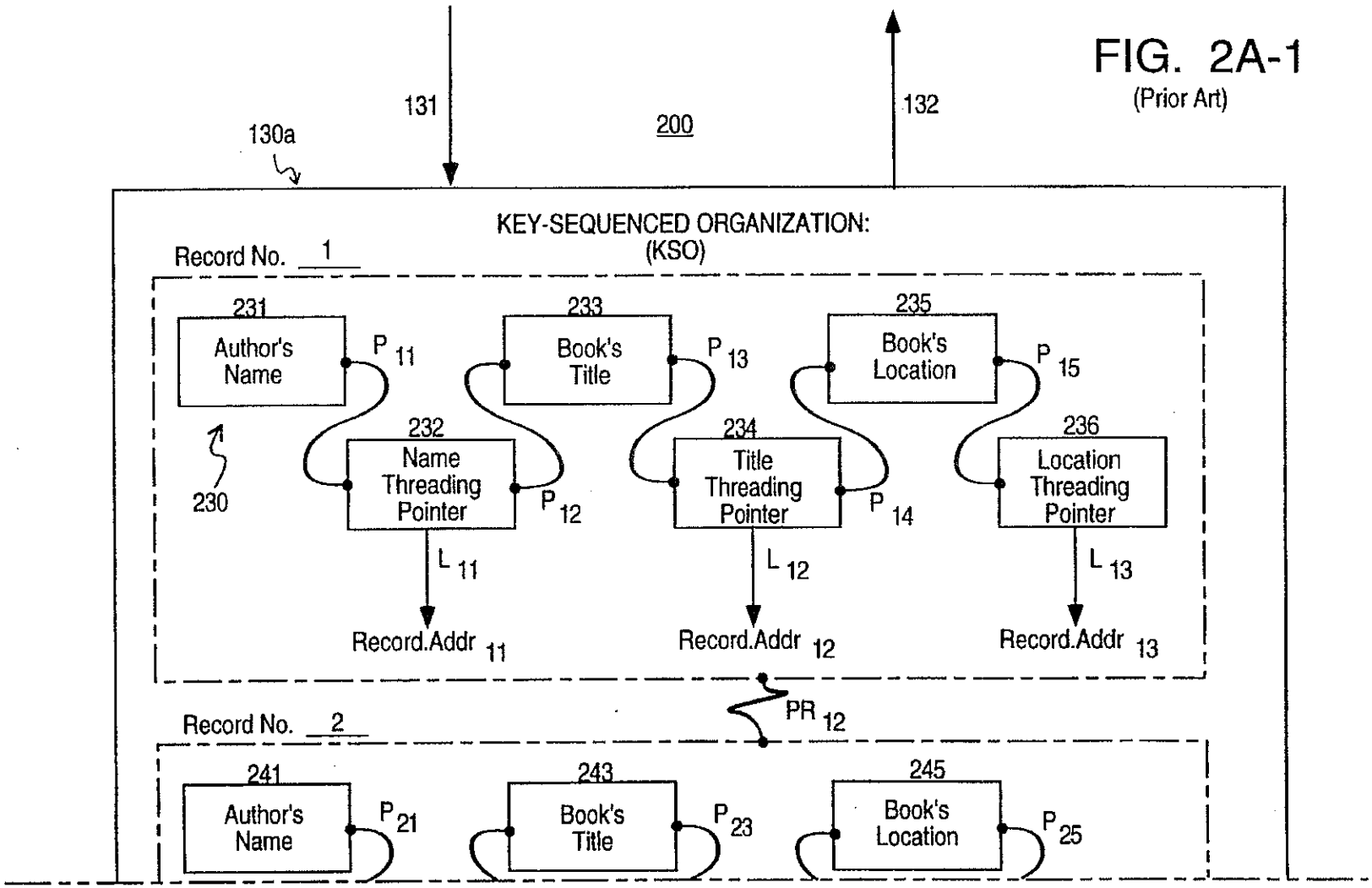



FIG. 1B
(Prior Art)

FIG. 2A-1
(Prior Art)



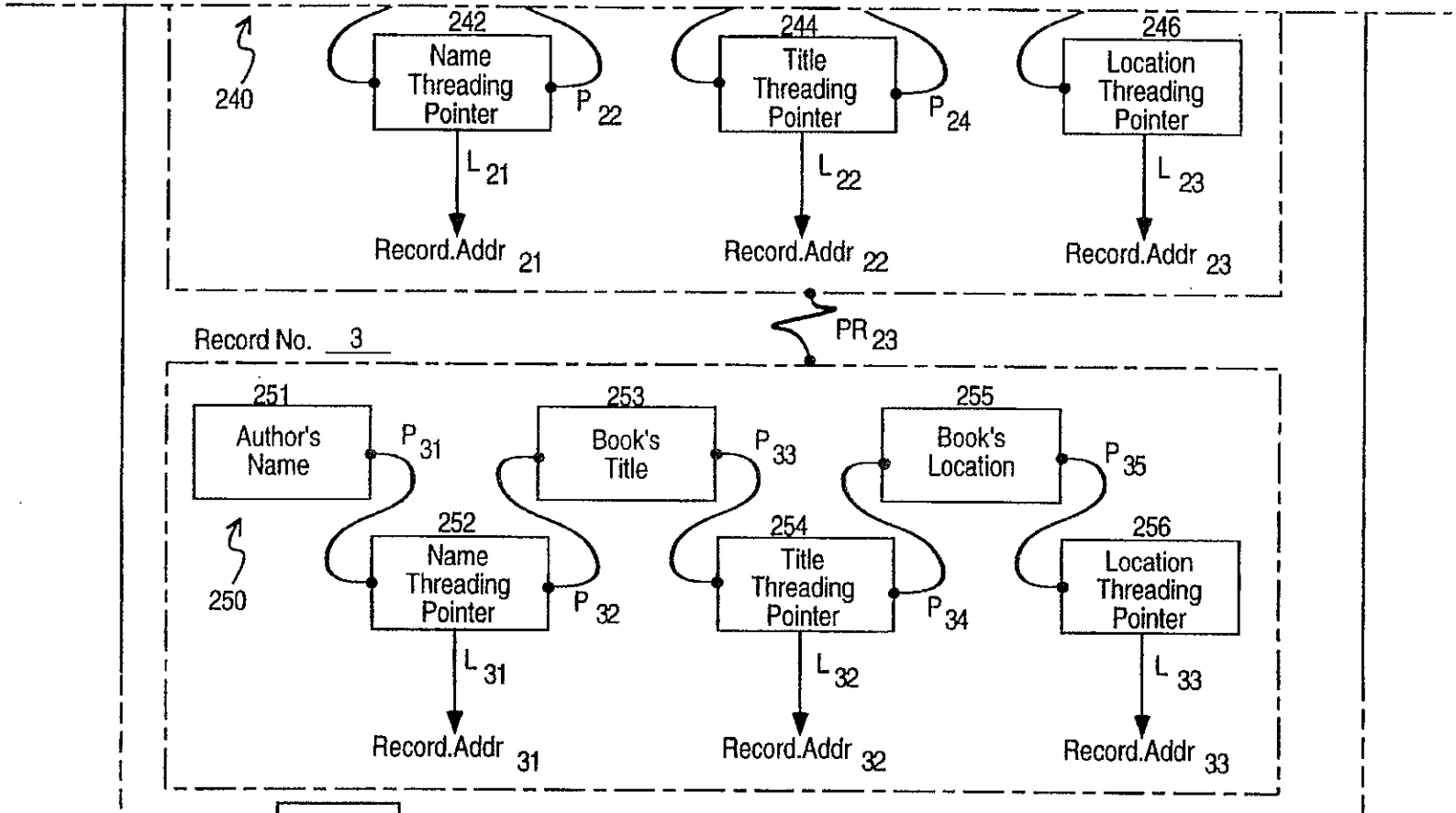
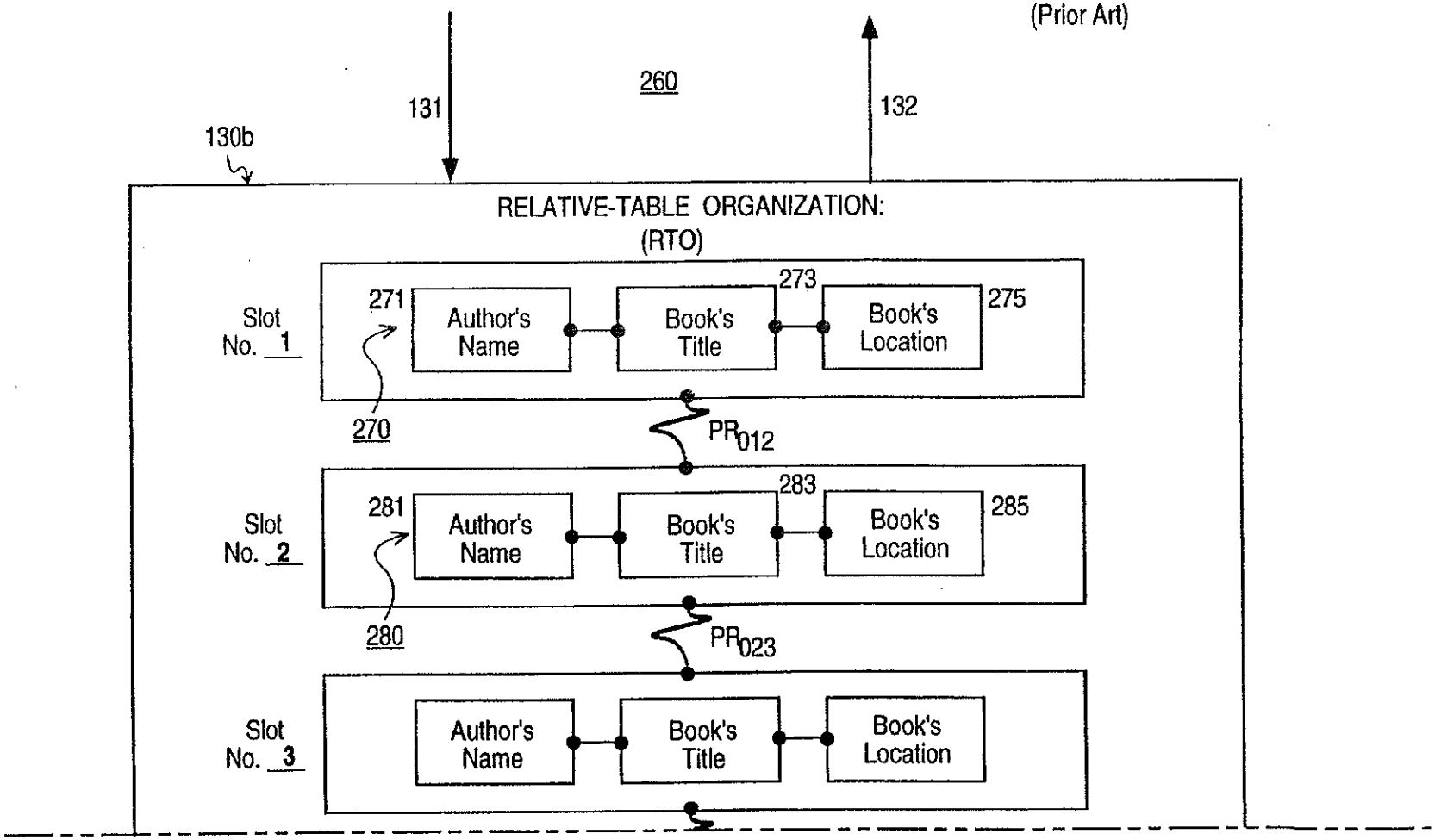



FIG. 2A-1
FIG. 2A-2
FIG. 2A

FIG. 2A-2
(Prior Art)

FIG. 2B-1
(Prior Art)



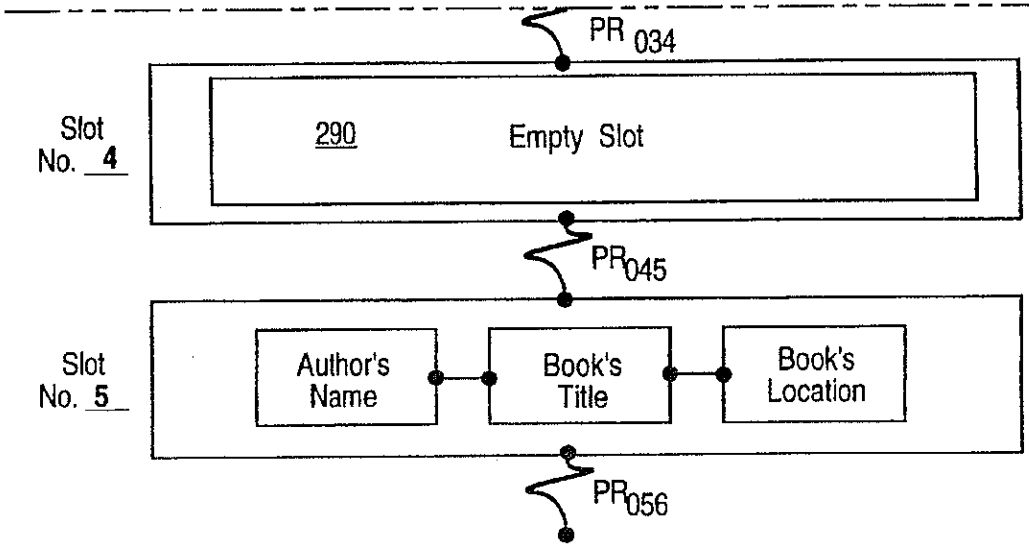



FIG. 2B-2
(Prior Art)

FIG. 2B-1
FIG. 2B-2
FIG. 2B

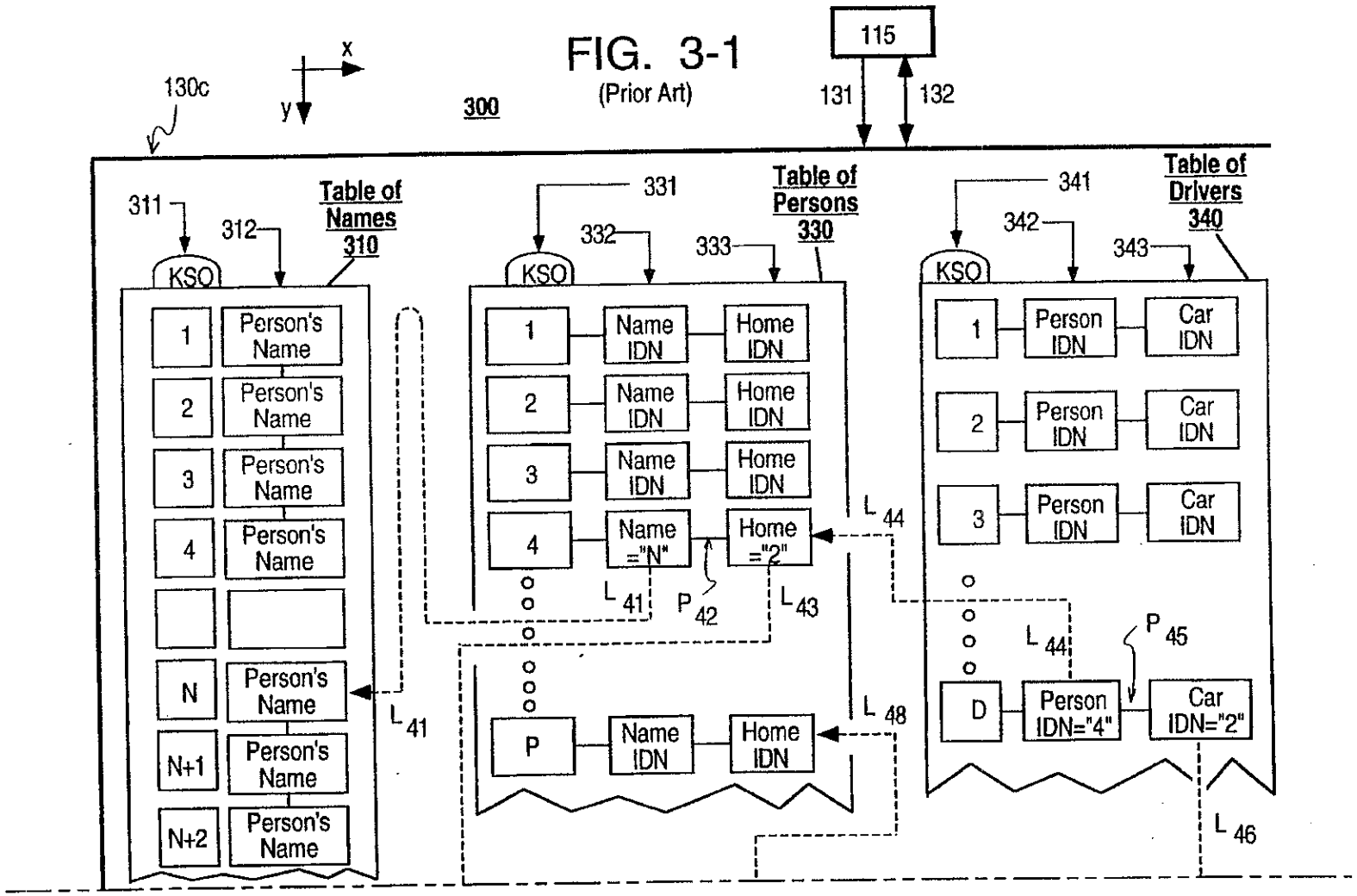



FIG. 3-1
(Prior Art)

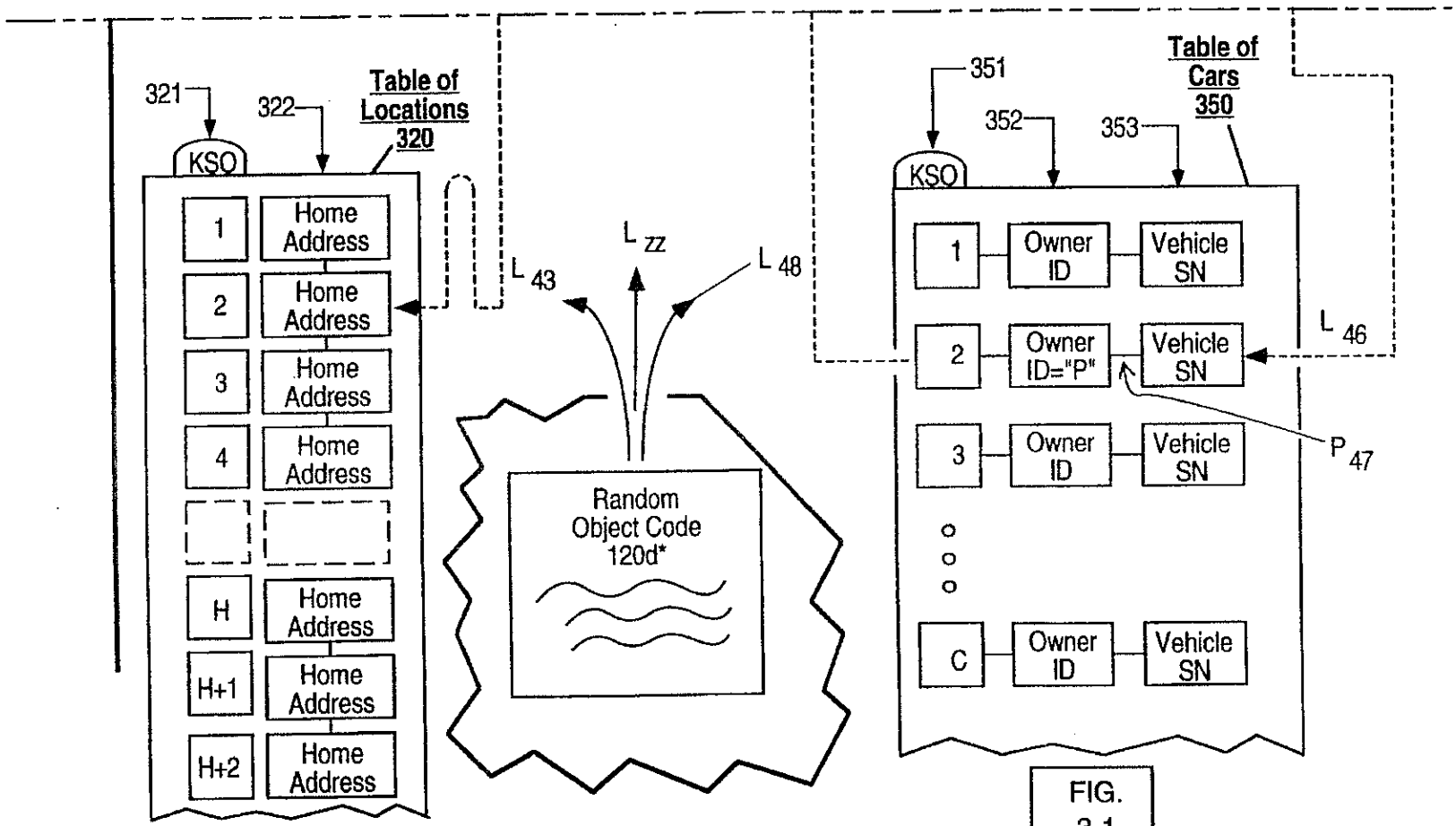



FIG. 3-2
(Prior Art)

FIG. 3-1
FIG. 3-2
FIG. 3

FIG. 4A

400

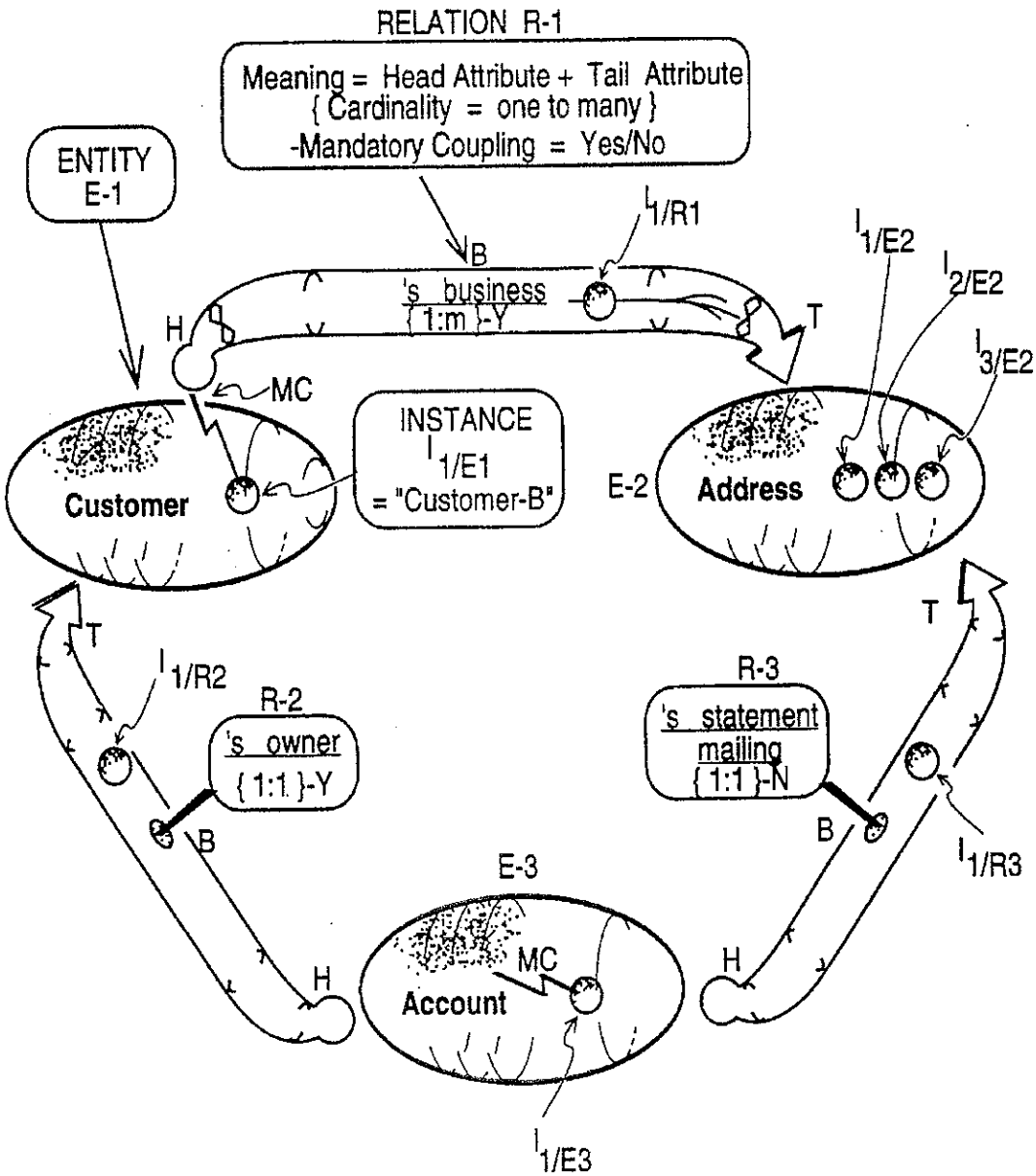

RELATION R-1
Meaning = Head Attribute + Tail Attribute
{ Cardinality = one to many }
-Mandatory Coupling = Yes/No
ENTITY E-1
I 1/R1
B
's business
{ 1:m }-Y
H
T
MC
I 1/E2
I 2/E2
I 3/E2
INSTANCE
I 1/E1
= "Customer-B"
Customer
E-2
Address
T
I 1/R2
R-2
's owner
{ 1:1 }-Y
B
H
R-3
's statement
mailing
{ 1:1 }-N
T
B
I 1/R3
E-3
MC
Account
H
I 1/E3

FIG. 4B

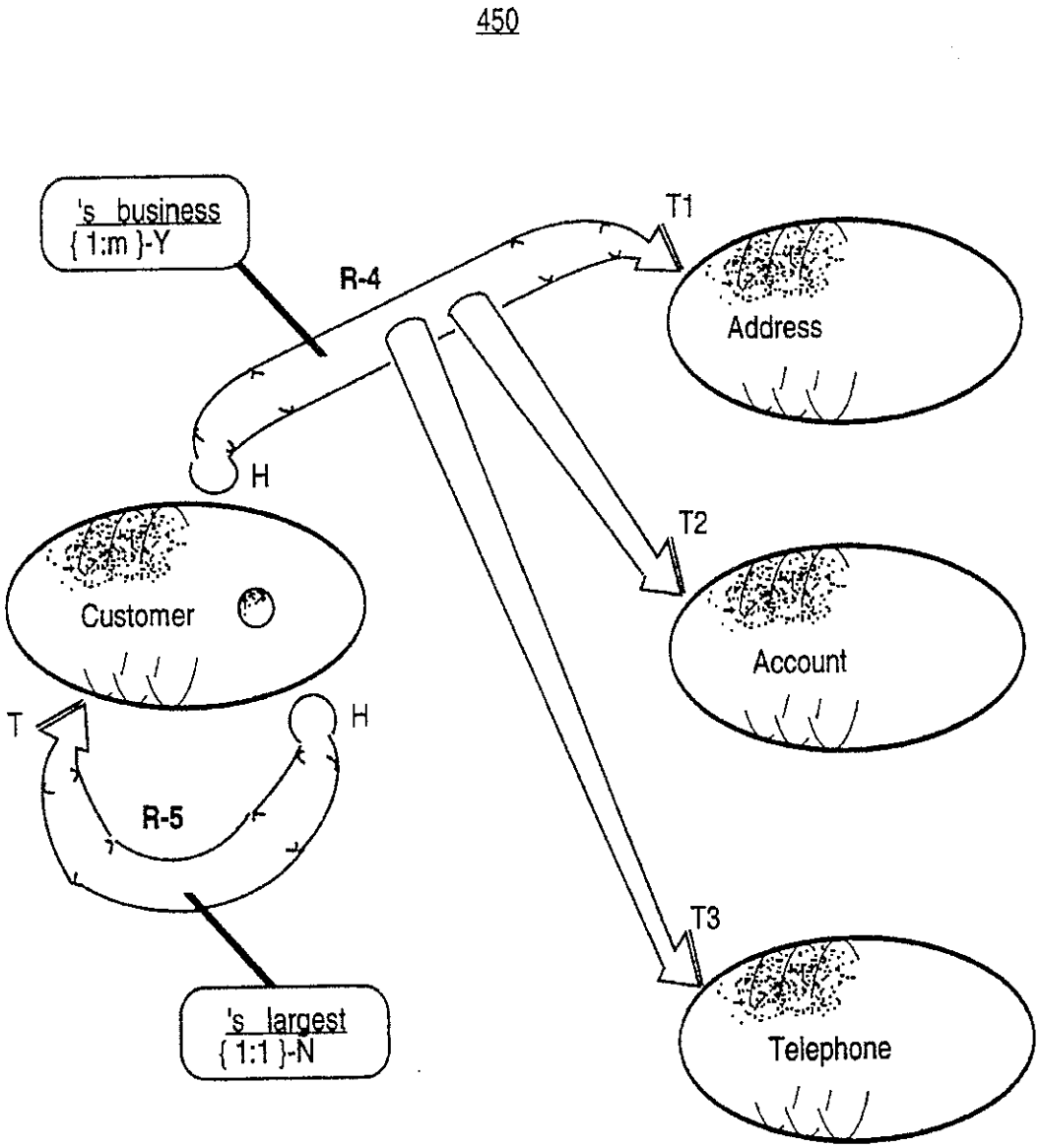

450
's business
{ 1:m }-Y
R-4
T1
Address
H
T2
Customer
Account
T
H
R-5
T3
's largest
{ 1:1 }-N
Telephone

FIG. 5

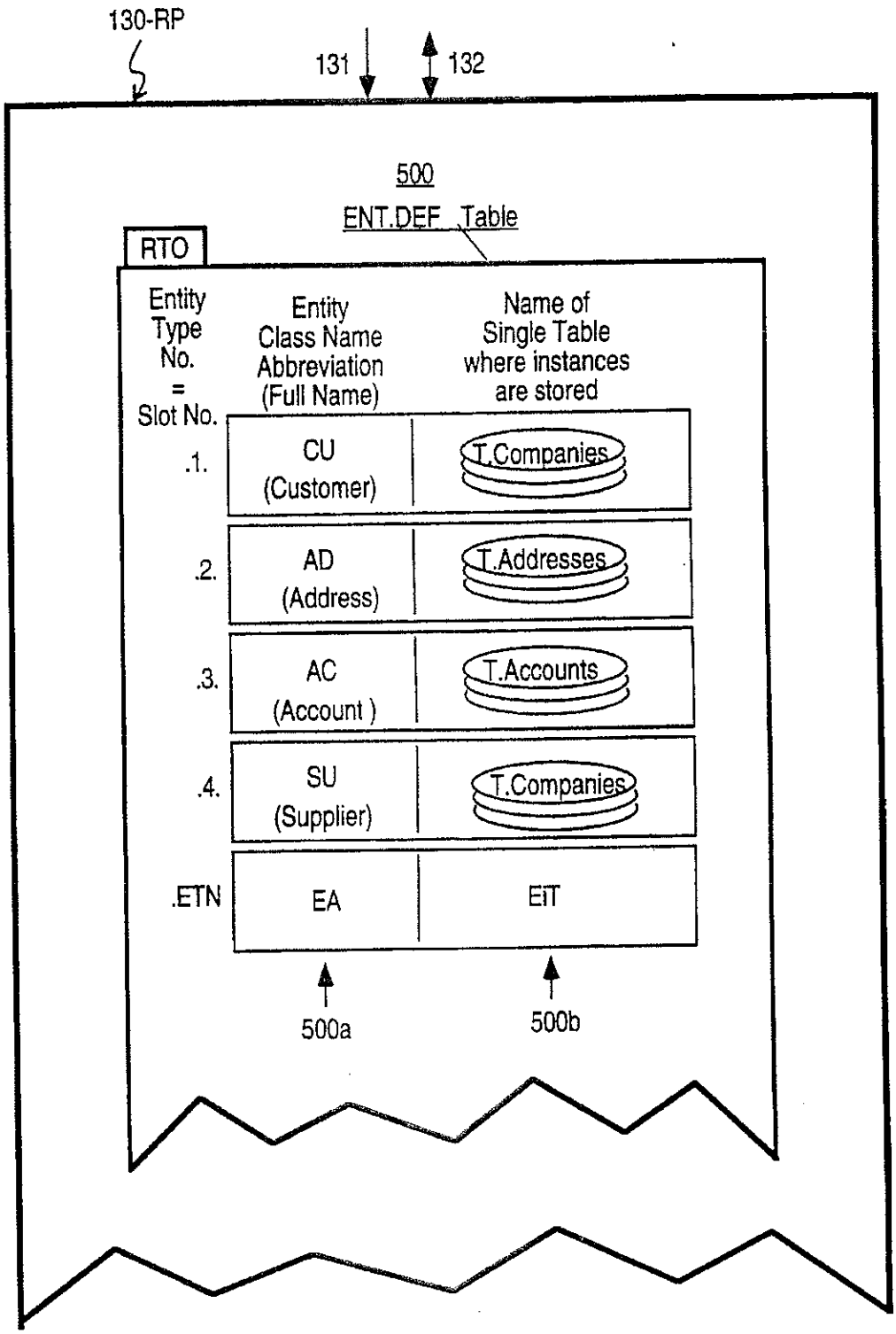

130-RP
131
132
500
ENT.DEF Table
RTO
Entity Type No. = Slot No.
Entity Class Name Abbreviation (Full Name)
Name of Single Table where instances are stored
.1.
CU (Customer)
T.Companies
.2.
AD (Address)
T.Addresses
.3.
AC (Account )
T.Accounts
.4.
SU (Supplier)
T.Companies
.ETN
EA
EiT
500a
500b

FIG. 6A

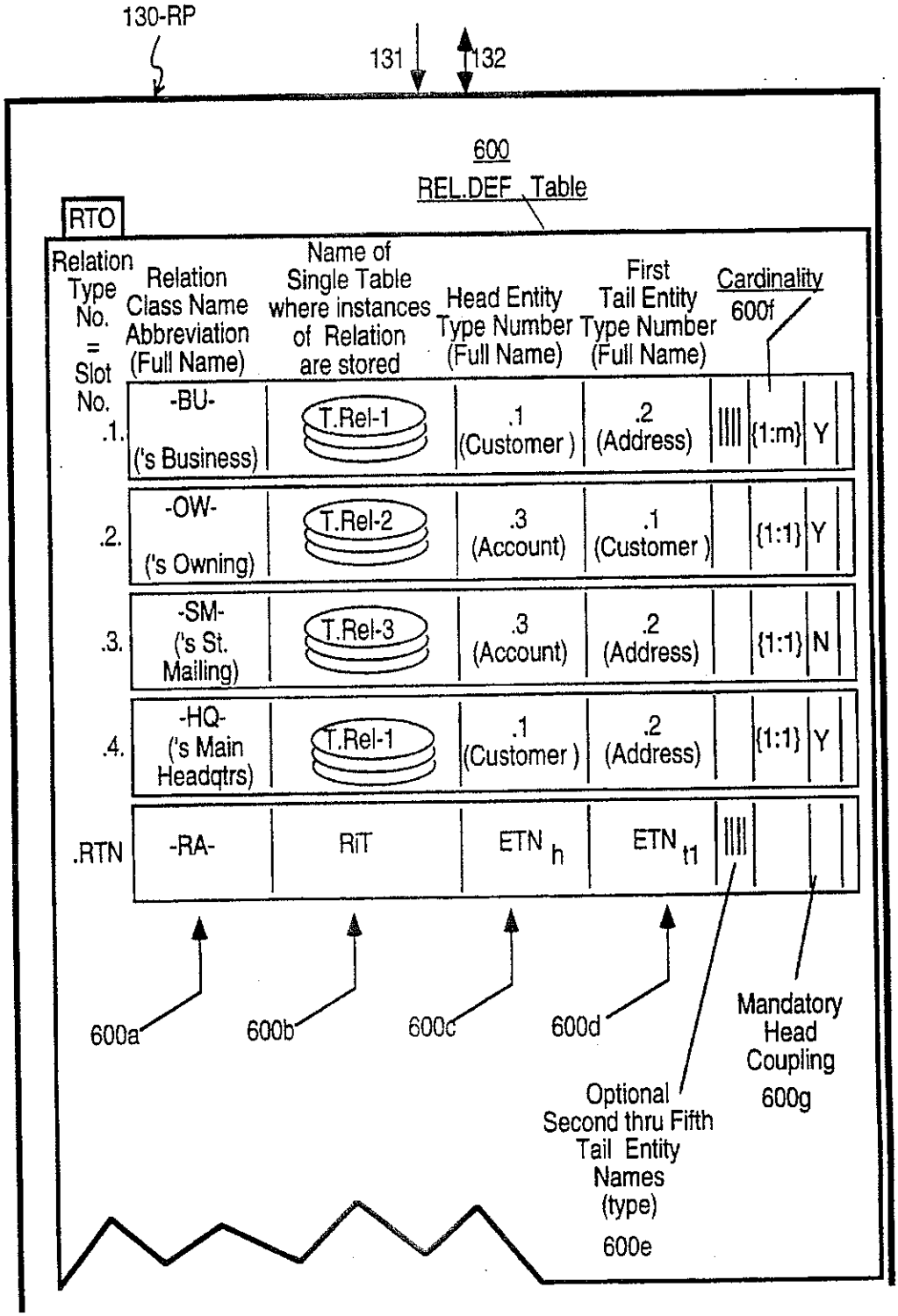

130-RP
131
132
600
REL.DEF. Table
RTO
Relation Type No. = Slot No.
Relation Class Name Abbreviation (Full Name)
Name of Single Table where instances of Relation are stored
Head Entity Type Number (Full Name)
First Tail Entity Type Number (Full Name)
Cardinality
600f
.1. -BU- ('s Business) T.Rel-1 .1 (Customer) .2 (Address) {1:m} Y
.2. -OW- ('s Owning) T.Rel-2 .3 (Account) .1 (Customer) {1:1} Y
.3. -SM- ('s St. Mailing) T.Rel-3 .3 (Account) .2 (Address) {1:1} N
.4. -HQ- ('s Main Headqtrs) T.Rel-1 .1 (Customer) .2 (Address) {1:1} Y
.RTN -RA- RiT ETN h ETN t1
600a
600b
600c
600d
Mandatory Head Coupling
600g
Optional Second thru Fifth Tail Entity Names (type)
600e

FIG. 6B

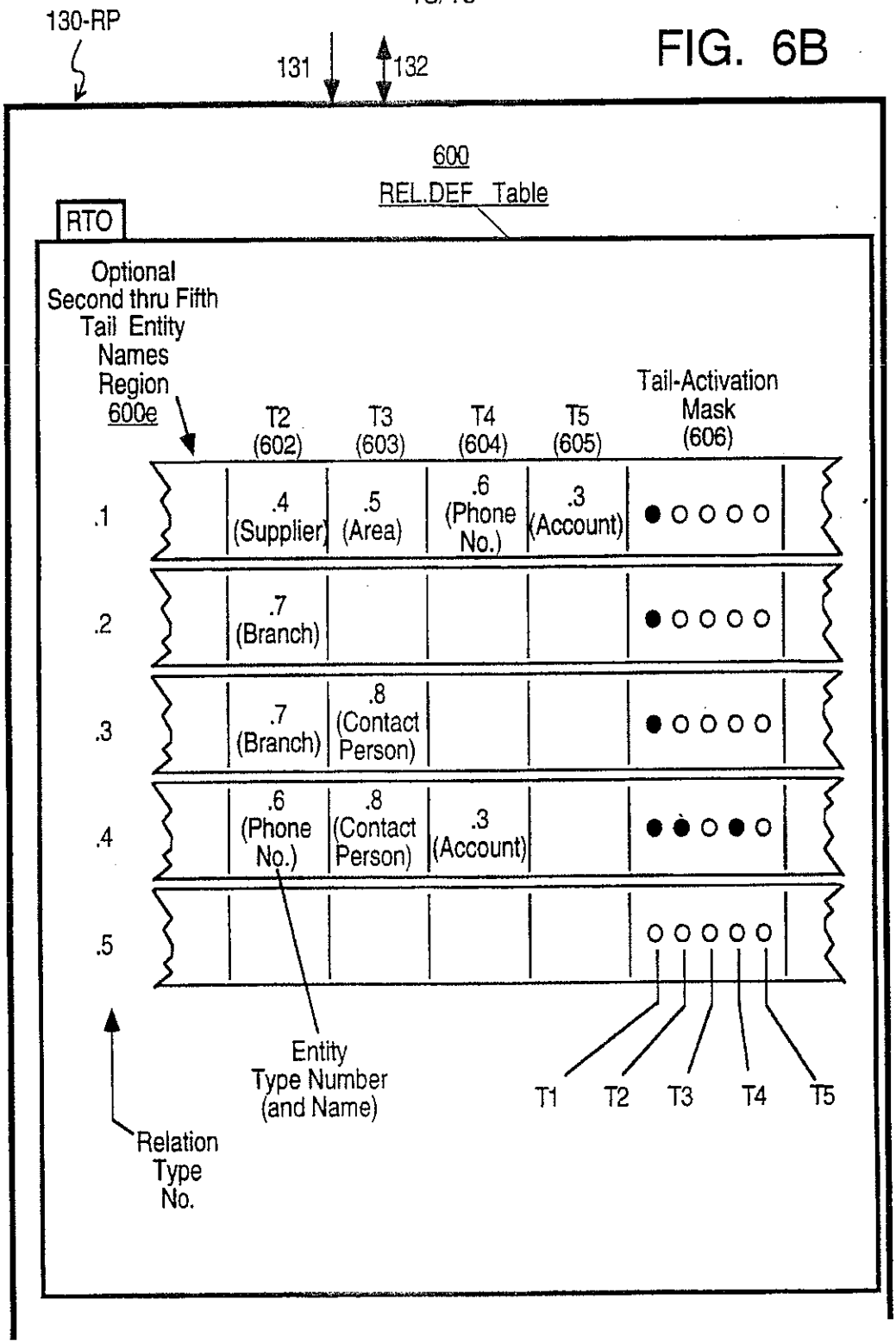
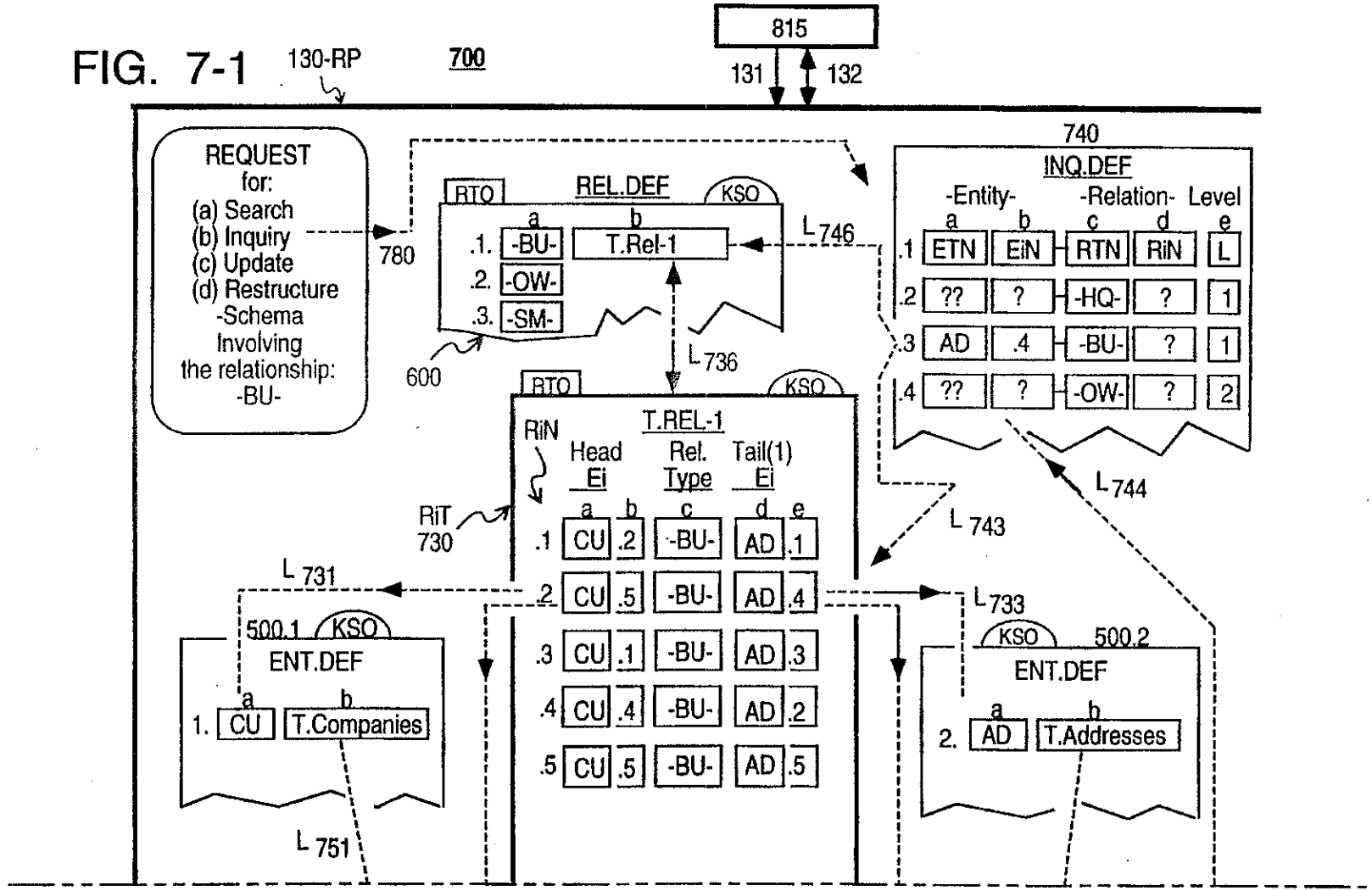

130-RP
131
132
600
REL.DEF Table
RTO
Optional Second thru Fifth Tail Entity Names Region 600e
T2 (602)
T3 (603)
T4 (604)
T5 (605)
Tail-Activation Mask (606)
.1
.4 (Supplier)
.5 (Area)
.6 (Phone No.)
.3 (Account)
.2
.7 (Branch)
.3
.7 (Branch)
.8 (Contact Person)
.4
.6 (Phone No.)
.8 (Contact Person)
.3 (Account)
.5
Entity Type Number (and Name)
Relation Type No.
T1
T2
T3
T4
T5

FIG. 7-1


130-RP
700
815
131
132
REQUEST
for:
(a) Search
(b) Inquiry
(c) Update
(d) Restructure
-Schema
Involving
the relationship:
-BU-
780
600
RTO
REL.DEF
KSO
a
b
.1. -BU- T.Rel-1
.2. -OW-
.3. -SM-
L 746
L 736
740
INQ.DEF
-Entity-
-Relation-
Level
a b c d e
.1 ETN EiN RTN RiN L
.2 ?? ? -HQ- ? 1
.3 AD .4 -BU- ? 1
.4 ?? ? -OW- ? 2
L 744
L 743
RTO
KSO
T.REL-1
RiN
Head Ei
Rel. Type
Tail(1) Ei
RiT
730
a b c d e
.1 CU .2 -BU- AD .1
.2 CU .5 -BU- AD .4
.3 CU .1 -BU- AD .3
.4 CU .4 -BU- AD .2
.5 CU .5 -BU- AD .5
L 731
L 733
500.1
KSO
ENT.DEF
a b
1. CU T.Companies
L 751
KSO
500.2
ENT.DEF
a b
2. AD T.Addresses

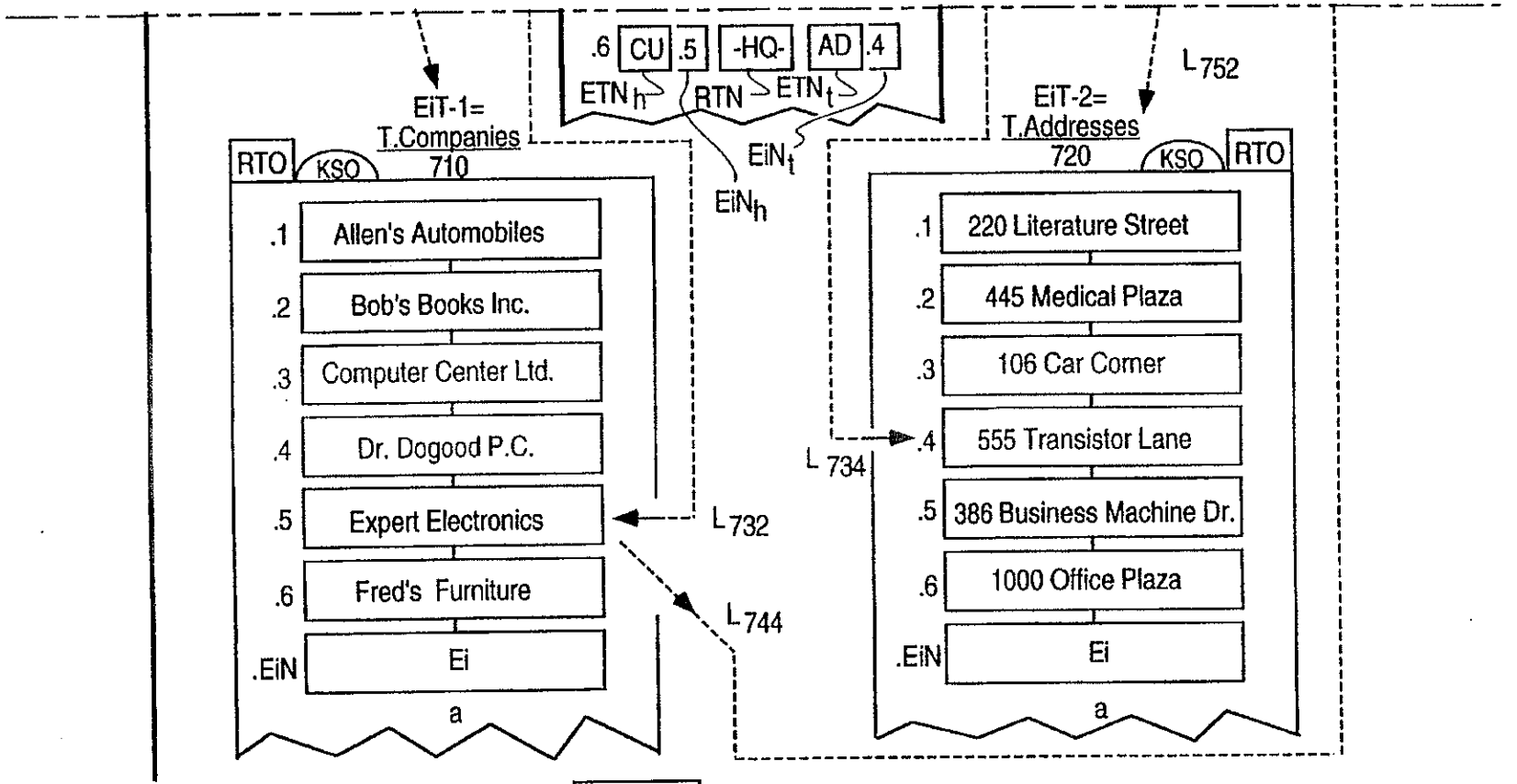



FIG. 7-2

| FIG. 7-1 |
| --- |
| FIG. 7-2 |

FIG. 7

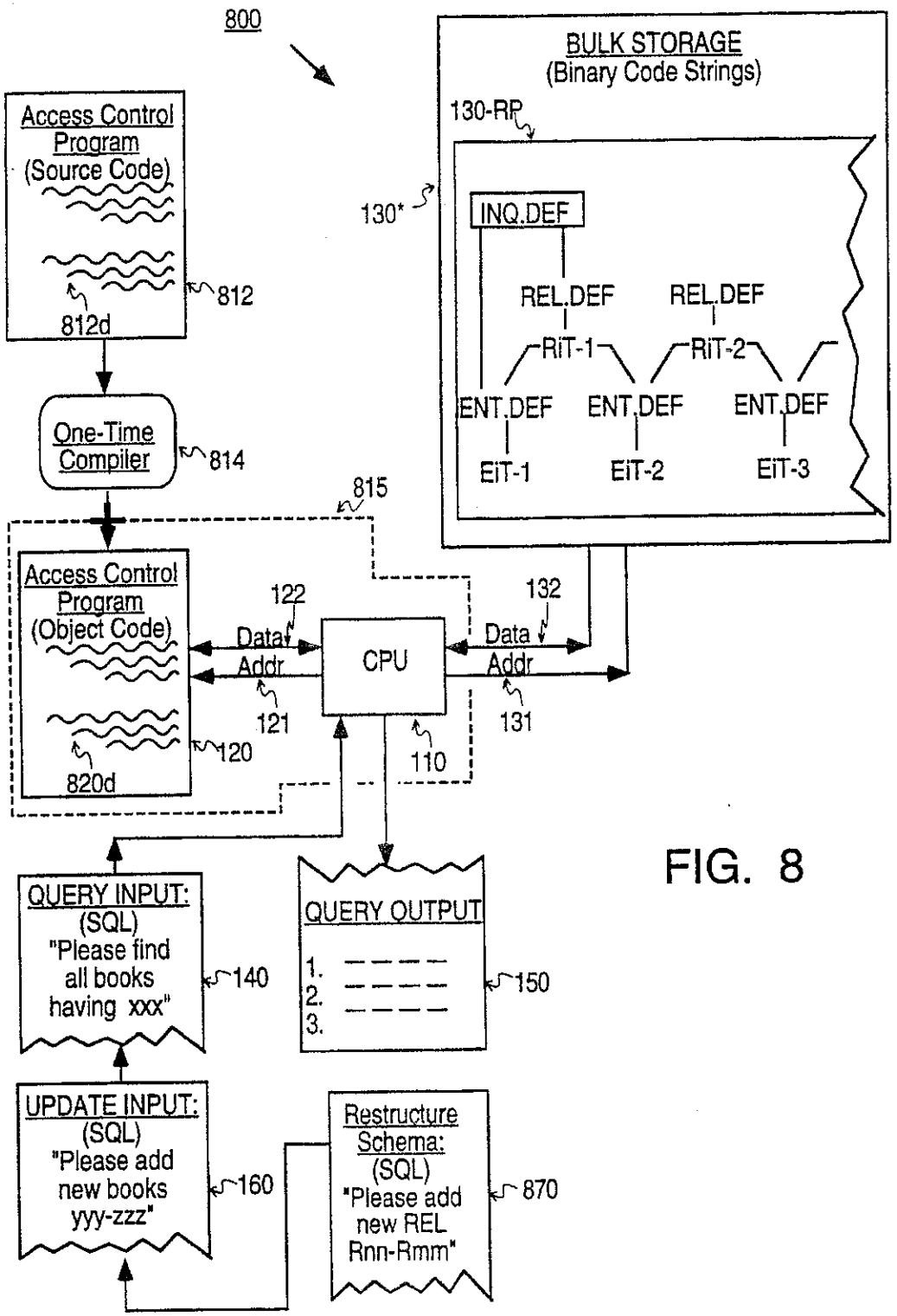

800
BULK STORAGE
(Binary Code Strings)
130-RP
130*
INQ.DEF
REL.DEF
REL.DEF
RiT-1
RiT-2
ENT.DEF
ENT.DEF
ENT.DEF
EiT-1
EiT-2
EiT-3
Access Control Program (Source Code)
812
812d
One-Time Compiler
814
815
Access Control Program (Object Code)
820d
120
122
Data
Addr
121
CPU
110
132
Data
Addr
131
QUERY INPUT: (SQL) "Please find all books having xxx"
140
QUERY OUTPUT
1.
2.
3.
150
UPDATE INPUT: (SQL) "Please add new books yyy-zzz"
160
Restructure Schema: (SQL) "Please add new REL Rnn-Rmm"
870

FIG. 8

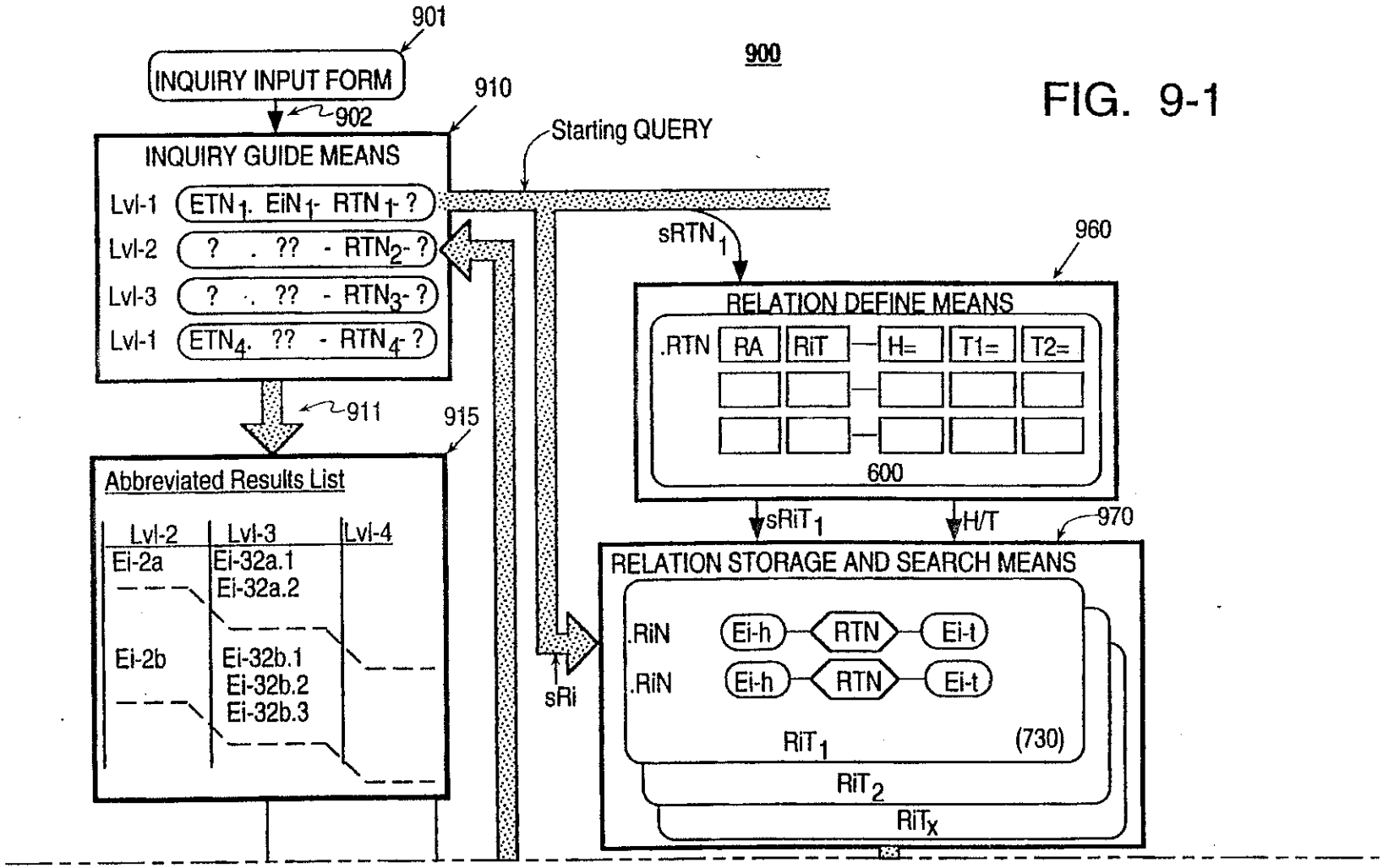

900
901
INQUIRY INPUT FORM
902
910
INQUIRY GUIDE MEANS
Lvl-1 ETN1. EiN1- RTN1- ?
Lvl-2 ? . ?? - RTN2- ?
Lvl-3 ? . ?? - RTN3- ?
Lvl-1 ETN4. ?? - RTN4- ?
Starting QUERY
911
915
Abbreviated Results List
Lvl-2 Lvl-3 Lvl-4
Ei-2a Ei-32a.1
Ei-32a.2
Ei-2b Ei-32b.1
Ei-32b.2
Ei-32b.3
sRi
sRTN1
960
RELATION DEFINE MEANS
.RTN RA RiT H= T1= T2=
600
sRiT1
H/T
970
RELATION STORAGE AND SEARCH MEANS
.RiN Ei-h RTN Ei-t
.RiN Ei-h RTN Ei-t
RiT1
(730)
RiT2
RiTx

FIG. 9-1

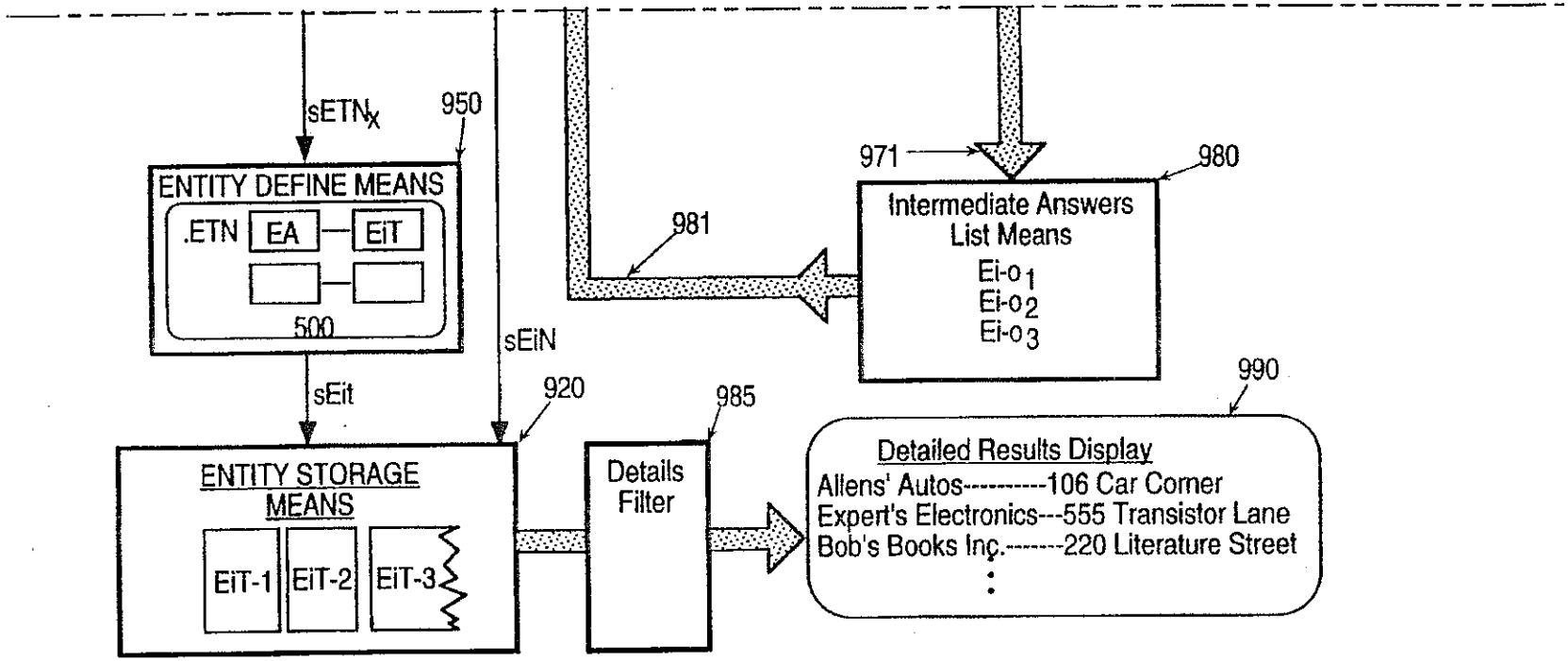



FIG. 9-1
FIG. 9-2
FIG. 9

FIG. 9-2

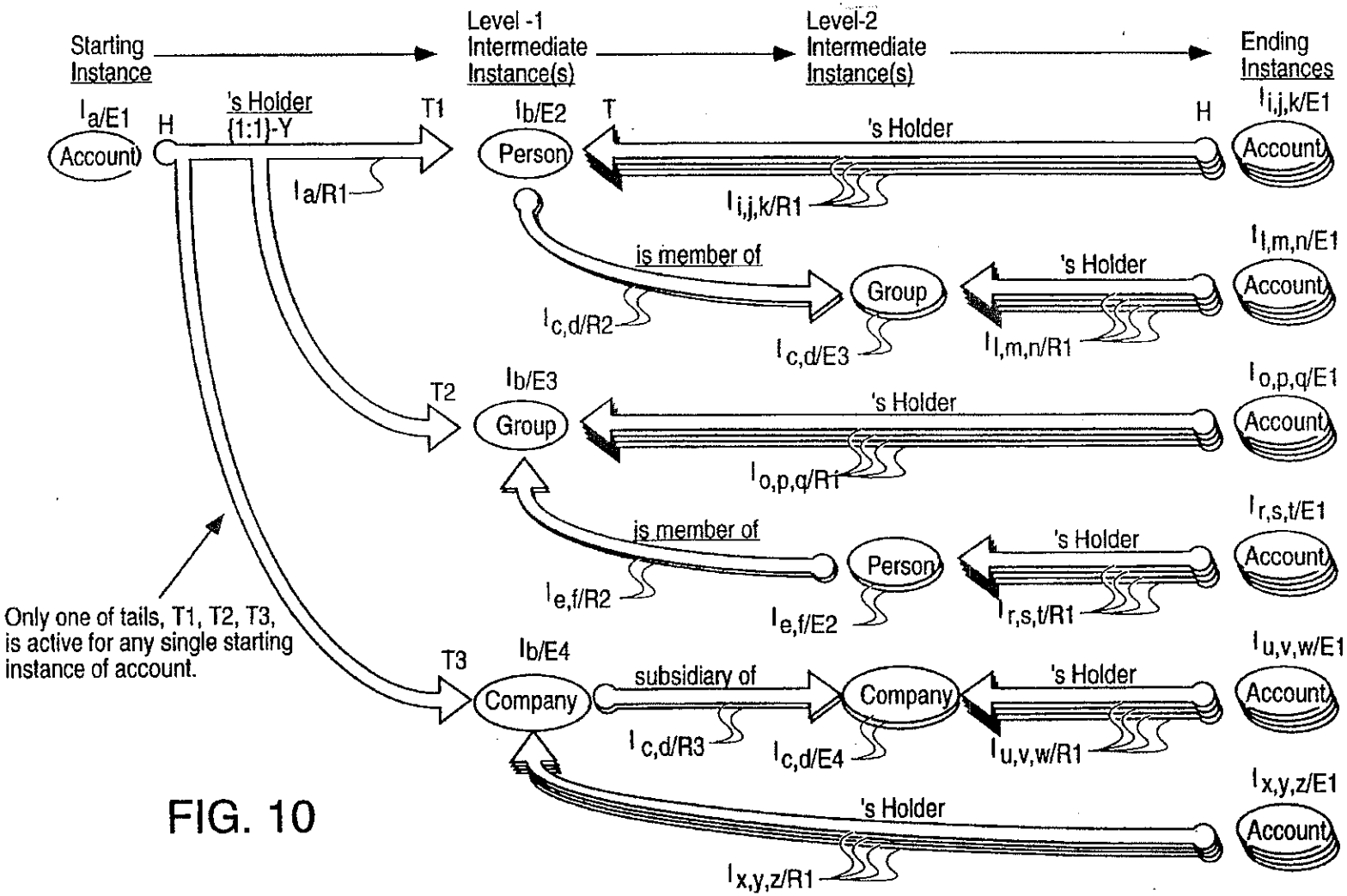

Starting Instance
Level -1 Intermediate Instance(s)
Level-2 Intermediate Instance(s)
Ending Instances
Ia/E1
Account
H
's Holder {1:1}-Y
T1
Ia/R1
Ib/E2
Person
T
's Holder
H
Ii,j,k/E1
Account
Ii,j,k/R1
is member of
Ic,d/R2
Group
Ic,d/E3
's Holder
Il,m,n/R1
Il,m,n/E1
Account
T2
Ib/E3
Group
's Holder
Io,p,q/R1
Io,p,q/E1
Account
is member of
Ie,f/R2
Person
Ie,f/E2
's Holder
Ir,s,t/R1
Ir,s,t/E1
Account
Only one of tails, T1, T2, T3, is active for any single starting instance of account.
T3
Ib/E4
Company
subsidiary of
Ic,d/R3
Company
Ic,d/E4
's Holder
Iu,v,w/R1
Iu,v,w/E1
Account
's Holder
Ix,y,z/R1
Ix,y,z/E1
Account

FIG. 10

# DATA PROCESSING SYSTEM AND METHOD FOR RETRIEVING AND ENTITY SPECIFIED IN A SEARCH PATH RECORD FROM A RELATIONAL DATABASE

This application is a division of application Ser. No. 08/083,361, filed Jun. 28, 1993 now abandoned, which is a continuation of Ser. No. 07/526,424, filed May 21, 1990, now abandoned.

## BACKGROUND OF THE INVENTION

1. Cross Reference to Microfiche Appendix

This application includes a plurality of computer program listings (modules) in the form of a Microfiche Appendix which is being filed concurrently herewith as 1162 frames (not counting target and title frames) distributed over 20 sheets of microfiche in accordance with 37 C.F.R. §1.96. The disclosed computer program listings are incorporated into this specification by reference but it should be noted that the source code and/or the resultant object code of the disclosed program modules are subject to copyright protection, The copyright owner has no objection to the facsimile reproduction by anyone of the patent document (or the patent disclosure as it appears in the files or records of the U.S. Patent and Trademark Office) for the sole purpose of studying the disclosure but otherwise reserves all other rights to the disclosed computer program modules including the right to reproduce said computer program modules in machine-executable form.

2. Field of Invention

The present invention relates generally to computer database management systems and more specifically to apparatus and methods for modifying and searching through large scale databases at high speed. 3. Description of Related Art

Modern computer systems are capable of storing voluminous amounts of information in bulk storage means such as magnetic disk banks. The volume of stored information can be many times that of the textual information stored in a conventional encyclopedia or in the telephone directory of a large city. Moreover, modern computer systems can sift through the contents of their bulk storage means at extremely high speed, accessing as many as one million bytes of information or more per second (a byte is a string of eight bits, equivalent to approximately one character of text in layman's terms). Despite this capability, it may take an undesirably long time (i.e., hours or days) to retrieve desired pieces of information. In commercial settings such as financial data storage facilities, there will be literally billions of pieces of information that could be sifted through before the right one or more pieces of information are found. Thus, even at speeds of one million examinations per second, it can take thousands of seconds (many hours) to retrieve a desired piece of information. Efficient organization of the stored information is needed in order to minimize retrieval time.

The methods by which pieces of information are organized within a computer, searched through or reorganized, often parallel techniques used by older types of manual information processing systems. A well known example of a manual system is the index card catalog found in public libraries. Such a card catalog consists of a large number of uniformly dimensioned paper cards which are serially stacked in one or more trays. The cards are physically positioned such that each card is directly adjacent to no more than two others (for each typical examination there is a



preceding card, the card under examination and a following card in the stack). On the front surface of each index card a librarian enters, in left to right sequence; the last name of an author, the first name of the author, the title of a single book which the author wrote and a shelf number indicating the physical location within the library where the one book may be found. Each of these four entries may be referred to as a "column" entry. Sufficient surface area must be available on each card to contain the largest of conceivable entries.

After the entries are made, the index cards are stacked one after the next in alphabetical order, according to the author's last name and then according to the author's first name and then by title. This defines a "key-sequenced" type of database whose primary sort key is the author's name. The examination position of each card is defined relative to the contents of preceding and following cards in the stack. That is, when cards are examined, each intermediate card is examined immediately after its alphabetically preceding card and immediately before its alphabetically succeeding card. When a new book is acquired, the key-sequenced database is easily "updated" by inserting a new card between two previously created cards. Similarly, if a book is removed from the collection, its card is simply pulled from the card stack to reflect the change.

If a library user has an inquiry respecting the location of a particular book or the titles of several books written by a named author, the librarian may quickly search through the alphabetically ordered set of index cards and retrieve the requested information. However, if a library user has an inquiry which is not keyed to an author's name, the search and retrieval process can require substantially more time; the worst case scenario being that for each inquiry the librarian has to physically sift through and examine each card in the entire catalog. As an example of such a scenario, suppose that an inquiring reader asks for all books in the library where the author's first name is John and the title of the book contains the word "neighbor" or a synonym thereof. Although it is conceptually possible to answer this inquiry using the information within the catalog, the time for such a search may be impractically long, and hence, while the information is theoretically available, it is not realistically accessible.

To handle the more common types of inquiries, libraries often keep redundant sets of index cards. One set of cards is sorted according to author names and another set is sorted according to the subject matter of each book. This form of redundant storage is disadvantageous because the size of the card catalog is doubled and hence, the cost of information storage is doubled. Also, because two index cards must be generated for each new book added to the collection the cost of updating the catalog is also doubled.

The size of a library collection tends to grow over time as more and more books are acquired. During the same time, more and more index cards are added to the catalog. The resulting stack of cards, which may be viewed as a kind of "database", therefore grows both in size and in worth. The "worth" of the card-based system may be defined in part as the accumulated cost of all work that is expended in creating each new index card and in inserting the card into an appropriate spot in the stack.

As time goes by, not only does the worth and size of the database grow, but new technologies, new rules, new services, etc., begin to emerge and the information requirements placed on the system change. Some of these changes may call for a radical reorganization of the card catalog system. In such cases, a great deal of work previously

expended to create the catalog system may have to be discarded and replaced with new work.

For the sake of example, let it be supposed that the library acquires a new microfilm machine which stores copies of a large number of autobiographies. The autobiographies discuss the life and literary works of many authors whose books are kept in the library. Let it further be supposed that the original, first card catalog system is now required to cross reference each book to the microfilm location (or plural locations) of its author's (or plural authors') autobiographies. In such a case, the card catalog system needs to be modified by adding at least one additional column of information to each index card to indicate the microfilm storage locations of the relevant one or more autobiographies.

We will assume here that there is not enough surface area available on the current index cards for adding the new information. Larger cards are therefore purchased, the information from the old cards is copied to the new cards, and finally, the new microfilm cross referencing information is added to the larger cards. This type of activity will be referred to here as "restructuring" the database.

Now let us suppose, that as more time goes by, an additional but previously unanticipated, cross indexing category is required because of the introduction of a newer technology or a new government regulation. It might be that the just revised and enlarged second card system does not have the capacity to handle the demands of the newer technology or regulation. In such a situation, a third card system has to be constructed from scratch. The value of work put into the creation of the just-revised second system is lost. As more time passes and further changes emerge in technology, regulations, etc., it is possible that more major organizational changes will have to be made to the catalog system. Time after time, a system will be built up only to be later scrapped because it fails to anticipate a new type of information storage and retrieval operation. This is quite wasteful.

Although computerized database systems are in many ways different from manual systems, the computerized information storage and retrieval systems of the prior art are analogous to manual systems in that the computerized databases require similar restructuring every time a new category of information relationships or a new type of inquiry is created.

At a fundamental level, separate pieces of information are stored within a computerized database system as a large number of relatively short strings of binary bits where each string has finite length. The bit strings are distributed spacially within a tangible medium of data storage such as an array of magnetic disks, optical devices or other information representing means capable of providing mass storage. Each bit is represented by a magnetic flux reversal, an optical perturbation and/or some other variance in the physical attributes of a data storage medium. A transducer or amplifier means converts these variances into signals (e.g., electrical, magnetic, or optical) which can be processed on a digital data processing machine. Each string of bits is often uniquely identified by its physical location or by a logical storage address. Some bit strings may function as address pointers, rather than as the final pieces of "real" information which a database user wishes to obtain. The address pointers are used to create so-called "threaded list" organizations of data wherein logical links between a first informational "object" (first piece of real data) and a second informational "object" (second piece of real data) are established by a chain of direct or indirect address pointers. The user-desired objects of real information themselves can be represented by a collection of one or more physically or logically connected strings.

Typically, "tables" of information are created within the mass storage means of the computerized system. A horizontal "row" of related objects, which is analogous to a single card in a card catalog system, may be defined by placing the corresponding bit strings of the objects in physical or address proximity with each other. Logical interconnections may be defined between different rows by using ancillary pointers (which are not considered here as the "real" data sought by a database user). A serial sequence of "rows" (analogous to a stack of cards) is then defined by linking one row to another according to a predefined sorting algorithm using threaded list techniques.

A vast number of different linking "threads" may be defined in this way through a database table having millions or billions of binary information bits. Unlike manual systems, the same collection of rows (which replaces the manual stack of cards) can be simultaneously ordered in many different ways by utilizing a multiplicity of threaded paths so that redundant data storage is not necessary. Searches and updates may be performed by following a prespecified thread from one row to the next until a sought piece of information (or its address) is found within a table. A threaded-list type of table can be "updated" in a manner similar to manual card systems by breaking open a logical thread within the list, at a desired point, and inserting a new row (card) or removing an obsolete row at the opened spot.

Tables are often constructed according to a "key-sequenced" approach. One column of a threaded-list table is designated as the sort-key column and the entries in that column are designated as "sort keys". Address pointers are used to link one row of the table to another row according to a predefined sequencing algorithm which orders the entries (sort-keys) of the sort column as desired (i.e., alphabetically, numerically or otherwise). Once a table is so sorted according to the entries of its sort column, it becomes a simple task to search down the sort column looking for an alphabetically, numerically or otherwise ordered piece of data. Other pieces of data which are located within the row of each sort key can then be examined in the same sequence that each sort key is examined. Any column can serve as the sort column and its entries as the sort keys. Thus a table having a large plurality of columns can be sorted according to a large number of sorting algorithms.

The key-sequencing method gives tremendous flexibility to a computerized database but not without a price. Each access to the memory location of a list-threading address pointer or to the memory location of a sort-key or to the memory area of "real" data which is located adjacent to a sort-key takes time. As more and more accesses are required to fetch pointers and keys leading to the memory location of a piece of sought-after information ("real data"), the response time to an inquiry increases and system performance suffers.

There is certain class of computerized databases which are referred to as "relational databases". Such database systems normally use threaded list techniques to define a plurality of key-sequenced "tables". Each table contains at least two columns. One column serves as the sort column while a second or further columns of the table store either the real data that is being sought or additional sort-key data which will ultimately lead to a sought-after piece of real data. The rows of the table are examined in an ordered fashion according to the contents of the sort column. Target

data is located by first threading down the sort column and thus moving through the chain of rows within a table according to a prespecified sort algorithm until a specific sort-key is found. Then the corresponding row is examined horizontally and the target data (real data or the next key) is extracted from that row.

An example of "real" data would be the full-legal names of unique persons such as in the character strings, "Mr. Harry W. Jones", "Mrs. Barbara R. Smith", etc. The sort-key can be a number which is stored adjacent to the full name and which sequences the names (real data) according to any of a wide variety of ordering patterns including by age, by height, by residential address, alphabetically, etc. Because the real data (e.g., full name of a person) is stored in a separate column, it is independent from the sort key data. A large variety of different relations can therefore be established between a first piece of real data (e.g., a first person's name) and a second piece of real data (e.g., a second person's name) simply by changing the sort keys that are stored in the separate sort column (e.g., who is older than whom, who is taller, etc.). Plural orderings of the real data can be obtained at one time by providing many columns in one table, by storing alternate keys in the columns and by choosing one or more of these columns as the primary sort key column.

Relational database systems often include tables that do not store real data in a column adjacent to their sort-key column, but rather store a secondary key number which directs a searcher to a row in another key-sequenced table where a matching key number is held together with either a piece of sought-after real data or yet another forward referencing key number (e.g., an entry which in effect says "find the row which holds key number x of yet another table for further details"). With this indirect key-sequenced approach, a large number of tables can be simultaneously updated by changing one entry in a "base" table.

Relational database tables are normally organized to create implied set and subset "relations" between their respective items of pre-stored information. The elements of the lowest level subsets are stored in base tables and higher level sets are built by defining, in other tables, combinations of keys which point to the base tables. The implied relations between elements cannot be discerned by simply inspecting the raw data of each table. Instead, relations are flushed out only with the aid of an access control program which determines in its randomly-distributed object code, which table to examine first and what column to look at before beginning to search down the table's column for a key number and, when that key number is found, what other column to look at for the real data or a next key number. Relations between various "entities" of a relational database are implied by the sequence in which the computer accesses them.

By way of a concrete example, consider a first relational table (Names-Table) which lists the names of a large number of people in telephone directory style. Each name (each separate item of real data) is paired to a unique key number and the rows of this Names-Table are sorted sequentially according to the key number. A second relational table may be provided in the database (Cars-Table) which lists automobile (vehicle) identification numbers (VIN) each paired in its row with a second key number. If the second key number is matched by a corresponding key number in the first table, then a relationship might be implied between the entries of the two separate tables (Names-Table and Cars-Table). The "implied" relationship might be one of an infinite set of possibilities. The relationship could be, for example, that the car listed in the second table is "owned" by the person whose name is found next to a matching key in the first table. On the other hand, it might be implied that the matched person in the first table "drives" the car, or "cleans" the car or has some other relation to the car. It is left to the access control program to define what the relationship is between entities in the first table and entities in the second table.

It can be seen that relational database systems offer users a great deal of flexibility since an infinite number of relations may be defined (implied). Economy in maintaining (updating) the database is also provided since a change to a base table propagates through all other tables which reference the base table. The access control program of the database system can include information-updating modules which, for example, change the key number in the second table (Cars-Table) whenever ownership of a car changes. If the name of the new owner is already in the first table (Names-Table), it does not have to be typed a second time into a new storage area and thus, extra work and storage redundancy are avoided. The vehicle identification number (VIN) remains unchanged. Minimal work is thus expended on updating the database.

Despite these advantages, relational database systems suffer from expandability and restructuring problems similar to those of the above-described manual system. Sometimes the rows within a particular table have to be altered to add additional columns. This is not easily done. Suppose for example, that a new government regulation came into being, mandating that vehicles are to always be identified not only by a vehicle identification number (VIN) but also by the name and location of the factory where the vehicle was assembled. If spare columns are not available in the Cars-Table, the entire database may have to be restructured to create extra room in the storage means (i.e. the disk bank) for adding the newly required columns. New key numbers will have to be entered into the new columns of each row (e.g., a new "factory of assembly" key number) and sorted in order to comply with the newly mandated regulation. New search and inquiry routines will have to be written for handling the newly structured tables.

In the past, much of this restructuring work was done by reprogramming the computer at the object code or source code level. This process relied heavily on an expert programming staff. It was time consuming, costly and prone to programming errors. Worst of all, it had to be redone time and again as new informational requirements emerged just after a last restructuring project was completed. There is a need in the industry for a database management system which provides quick responses to inquiries and which can also be continuously updated or restructured without reprogramming at the source or object code level.

## SUMMARY OF THE INVENTION

It is an objective of the present invention to provide a database system which is capable of storing voluminous amounts of information, sifting through the information at high speed, and is at the same time easily expandable or restructurable to take on new forms of entities and relationships.

In accordance with a first aspect of the invention, an entity definition table (ENT.DEF) is defined within the memory means of a computer system to store the name of an allowed entity type (class) and the name of a single other table (Entity-instances Table or "EiT" for short) where instances of the allowed entity type may be stored. A separate rela-

tionships definition table (REL.DEF) is defined in the memory means to list in each row of the table: (a) the name of an allowed relations type, (b) the name of a single Relation-instances Table (RiT) where instances of the allowed relationship type may be stored, (c) the name of a primary (head) entity type to which the relation type may apply and (d) the names of one or more secondary (tail) entity types to which the named relationship may apply. Each row of the Relation-instances Table (RiT) is provided with at least one primary pointer which points to the storage location of a first instance of the primary entity type and at least one secondary pointer which points to the storage location of a corresponding first instance of the secondary entity type. Each row of the Relation-instances Table (RiT) further includes a pointer to a relationship-defining row in the REL.DEF table. The pointer can be the name of an applicable relation type as recorded in the REL.DEF table. Relationships between instances of a primary entity and a secondary entity are thus expressly defined by entries in the Relation-instances Table (RiT). Adding new rows to this Relation-instances Table (RiT) allows for the addition of new relations. Adding new rows to the REL.DEF table allows for the creation of new classes (types) of relationships. Since relation-defining tables can be updated using a fixed set of update modules, reprogramming at the source or assembly level is not needed for restructuring the schema of the database.

## BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be described with reference to the following figures in which:

FIG. **1A** is a block diagram of a conventional database system.

FIG. **1B** is a timing diagram showing the delay between the addressing and the delivery of storage data.

FIG. **2A** is a block diagram of a conventional key-sequenced table organization.

FIG. **2B** is a block diagram of a conventional relative-record table organization.

FIG. **3** diagrams a multiple table system which is based on a conventional relational database approach and which has key-sequence organized tables.

FIG. **4A** is a conceptual diagram illustrating an entity-relation schema in accordance with the invention.

FIG. **4B** is a further conceptual diagram of an entity-relation schema according to the invention.

FIG. **5** is a block diagram of an entity definition (ENT.DEF) table in accordance with the invention.

FIGS. **6A** and **6B** are block diagrams of a relationship definition (REL.DEF) table in accordance with the invention.

FIG. **7** is a connection diagram showing how relations may be explicitly defined in a Relation-instances Table (RiT) so that unique relations between instances of a first entity class and instances of a second entity class can be identified.

FIG. **8** is a block diagram of a database system according to the invention.

FIG. **9** is a block diagram of a relations processing engine according to the invention.

FIG. **10** graphs a variety of sample inquiry paths that may be followed by the engine of FIG. **9**.

## DETAILED DESCRIPTION

The following includes a detailed description of the best mode or modes presently contemplated by the inventor for carrying out the invention. It is to be understood that these modes are merely exemplary of the invention. The detailed description is not intended to be taken in a limiting sense.

Referring to FIG. **1A**, the block diagram of a conventional database system **100** is shown. The database system **100** comprises a central processing unit (CPU) **110** which is operatively coupled so as to be controlled by an access control program (object code) **120***d* stored in a first memory means **120** (i.e., read-only-memory, ROM, or random access memory, RAM). The CPU **110** in combination with the first memory means **120** can be viewed as one or more machine means for performing functions specified by the object code **120***d*. The CPU **110** is further operatively coupled to access the data **130***d* of a "bulk storage" second memory means **130** also included in the database system **100**. Individual strings of digital information are represented by wiggled lines (e.g., **120***d*, **130***d*) in FIG. **1A**. The bulk storage means **130** typically takes the form of a large array of magnetic disk drives, tape drives, or other mass storage devices (e.g., arrays of Dynamic Random Access Memory [DRAM] chips). The first (control) memory means **120** usually takes the form of high speed RAM and/or ROM.

To access a particular string of data **130***d* stored within the bulk storage means **130**, the CPU **110** must provide a corresponding address signal **131***s* (FIG. **1B**) in the form of logic highs (H) and lows (L) to the bulk storage means **130** over an address bus **131**. As seen in the time versus logic-level graph of FIG. **1B**, the address signal **131***s* (usually an electrical signal) comprises a set of logic high and logic low levels (H and L) transmitted in a first time period $t_0$–$t_1$. There follows a second time period, $t_1$–$t_2$, which is often referred to as an "access delay", during which addressing circuits attempt to access the addressed memory location. Depending on whether a memory read or memory write operation is occurring, data signals **132***s* are then transferred over a data bus **132** (FIG. **1A**) from the addressed location within the bulk storage means **130** to the CPU **110** or vice versa during a following third time period, $t_2$–$t_3$.

Referring still to FIG. **1A**, the object code **120***d* of the access control program determines when and how the CPU **110** will access information **130***d* stored in the bulk storage means **130**. The CPU **110** issues address signals **121***s* (not shown) over an address bus **121** to the first memory means **120**, and in response, the first memory means **120** supplies instruction signals **122***s* (not shown) over a data bus **122** to the CPU **110**. Information signals **122***s* can be exchanged bidirectionally over data bus **122** between the CPU **110** and the first memory means **120**. FIG. **1B** may represent the timing relation between address signals **121***s* and first memory information signals **122***s* by replacing reference numerals **131***s* and **132***s* with **121***s* and **122***s*, respectively.

It should be understood that neither the object code **120***d* of the first memory means **120** nor the data code **130***d* of the mass storage means **130** is in human-readable form. A translation machine is needed to convert the binary bit strings of either memory means (**120** or **130**) into a form which might be understandable to an experienced computer programmer or to a lay computer user.

The object code **120***d* of the access control program is produced by first generating (e.g., manually writing and encoding) a source code listing **112** whose lines of information **112***d* are usually understandable only to a highly trained computer programmer. The source code listing **112** which is written in an assembly level or higher level language (e.g., C, COBOL, FORTRAN, PASCAL, etc.) is transformed into machine-readable form, and passed

through a first translation machine which may be referred to as a compiler (or assembler) means **114**. The compiler means **114** produces the machine-readable object code **120***d* according to instructions provided by a machine readable version of the source code listing **112**. After it is stored in the first memory means **120**, the object code **120***d* is expressed as machine detectable alternations (ones and zeroes) in a physical attribute (e.g., voltage) of the medium which makes up the first memory means **120**. In this form, the object code **120***d* is more readily convertible into data signals **122***s* which are understandable to the CPU **110** than into information which is understandable to a lay (non-programmer) person. It is highly improbable that a lay person will ever wish to access or understand or modify the object code **120***d* stored within the first memory means **120**.

The information strings **130***d* within the bulk storage means **130** are similarly expressed as alternations in the physical property of the storage medium making up the second memory means **130**. Some of the data strings **130***d* represent "real" data which a lay-user may wish to access while others of the strings **130***d* represent "ancillary" data such as sequencing keys, threading pointers or control codes which a lay-user is not interested in. The object code **120***d* of the control program defines which is which.

When "real" data is to be extracted from the data strings **130***d* within the bulk storage means **130**, read and understood by a lay person, a translation process similar to compilation (or more correctly de-compilation) needs to take place. Just like the compiler means **114** functions as a man-to-machine translator, the combination of the first memory means **120** and the CPU **110** defines a second man-to-machine search-and-translate machine **115** which is used to search through parts of the bulk stored data **130***d*, extract relevant pieces of "real" data and convert the extracted data from machine-readable form into human-readable form. The human-readable output of the second translation machine **115** may be produced in the form of a query output listing **150** (e.g., on paper or on a video screen) as indicated in FIG. 1A.

If a lay user (defined here as someone other than a person who is an expert programmer familiar with details of the source listing **112**) wishes to obtain useful ("real") information from the bulk storage means **130**, the lay user will normally supply a query input **140**, in a form dictated by a so-called "structured query language" (SQL) to the CPU **110**. (In the illustrated example the user inputs the request string "Please find all books having attribute xxx," where xxx could be the relations "author's last name =Jones".) The combination of the CPU **110** and first memory means **120** (which combination forms the second search-and-translate machine **115**) process this query input **140** and in response, produces a series of address signals **131***s* which are sent to the bulk storage means **130** and processes a series of data retrievals **132***s* which eventually lead to the production of a corresponding query output listing **150**. (In the example, it would be a listing of all books whose author's name is "Jones".) The access control program **120***d* is charged with the task of enabling various types of queries **140** and making sure that the queries do not violate basic rules of logic.

When the information **130***d* within the bulk storage means **130** needs to be updated, by for example adding new books, a similar exchange occurs between the translating machine **115** and a lay user. The lay user supplies an update input **160**, again as dictated by a pre-specified structured query language (SQL), and in response, the translating machine **115** rearranges the data **130***d* within the bulk storage means **130** to achieve the requested update.

Referring to FIG. **2A**, a first embodiment **200** of the data base system **100** will be described in more detail. FIG. **2A** schematically illustrates a section **130***a* of the bulk storage means **130** according to embodiment **200** wherein some of the stored data strings **130***d* are arranged to define a key-sequenced type of table. In a first record region (Record No. 1) of the table **130***a* there is provided a first continuous data string **230** which is subdivided to have a first string portion **231** representing an author's name (illustrated as the contents of a rectangular box), a second string portion **232** contiguous thereto for representing a name threading pointer (illustrated as a second rectangular box coupled to the first rectangular box by an address proximity link $P_{11}$), a third data string portion **233** representing the book's title (which is linked to the second portion **232** by proximity link $P_{12}$), a fourth subsection **234** representing a title threading pointer (linked to box **233** by address proximity link $P_{13}$), a fifth subsection **235** representing the book's location (linked to box **234** by proximity link $P_{14}$) and a sixth subsection **236** representing a location threading pointer (linked to box **235** by proximity link $P_{15}$).

The name threading pointer **232** is located directly adjacent to the author's name subsection **231** within the address space of Record No. **1**, as indicated by address proximity link $P_{11}$ and thus, there is an "implied" logical connection between the data contents of boxes **231** and **232**. The book's title subsection **233** is located directly adjacent to the name threading pointer **232** as indicated by address proximity link $P_{12}$. The combined, proximity linkage, $P_{11}$–$P_{12}$, "implies" a relationship between the contents of boxes **231** and **233**, namely that they apply to various attributes of a common book. This format repeats for data subportions **234–236**. Only boxes **231**, **233** and **235** contain "real" data which is useful to a lay person. The other boxes, **232**, **234** and **236** of Record No. **1** contain "ancillary" data which is useful to the search machine **115** but does not provide the kind of "real" information sought by an inquiring lay person.

The implied relations between the "real" data boxes, **231**, **233** and **235** of Record No. **1**, arise only after "meaning" is assigned to all the boxes **231–236**. Such "meaning" comes from the operation of the search-and-translation machine **115** (FIG. **1**). To understand this concept, assume that an automated "searching" machine (computer) **115/200** of embodiment **200** is examining the data string **230** held within the single Record No. **1**. Assume further that this searching machine **115/200** includes means for assigning appropriate "meanings" to each of the data subportions contained in each of subsections **231–236** to thereby designate some as containing "real" data and others as containing "ancillary" (e.g., pointer) data. In that case the search machine **115/200** can scan horizontally across the record, parse the data string **230** into subsections of appropriate size and extract the name of the book's author, the book's title and the location of the book within the library, as desired. On the other hand, if the searching machine **115/200** does not possess information which tells it that box **232** is a threading pointer, box **233** is a title, etc., then all boxes will look alike to the search machine, there will be no "meaning" assigned and the search machine **115/200** will not be able to extract a desired piece of data. Thus, while not shown in FIG. **2A**, it is to be understood that there is a cooperative relation between how the object code **120***d* of the search machine **115/200** causes that search machine to access the parts of bit string **230** via the signal busses, **131** and **132**, how subportions of bit string **230** become designated as "real" or "ancillary" data, and how relations are implied between separate pieces of real data. The structure, meanings inter-

relations between the parts of bit string **230** are intimately linked to the structuring of the object code **120***d*.

In FIG. **2**A, the bulk memory means section **130***a* is shown to include additional record areas (Record No. **2**, Record No. **3**, etc.) each having the same data structure (represented respectively as string **240** which comprises data subsections **241–246** and string **250** which comprises data subsections **251–256**). Although Record No. 1 is in physical proximity with Record No. **2**, as indicated by physical (or address) proximity link $PR_{12}$, and Record No. **2** is in physical proximity with Record No. **3** as indicated by physical proximity link $PR_{23}$, the data items (**231–236**, **241–246**, **251–256**) within each record do not need to be examined according to this physical ordering. Instead, the name threading pointer **232** of Record No. 1 can represent the address of any other arbitrary record area within the bulk storage means section **130***a* whose author's-name will serially follow the author's-name of box **231** during a search process. This is represented in FIG. **2**A by the dashed logical link $L_{11}$ which points to some arbitrary record area, Record.Addr.$_{11}$ of section **130***a*. The name threading pointer of the referenced record, Record.Addr.$_{11}$, can point to yet another arbitrary record. With this mechanism, a list which is sorted (alphabetically for example) according to author's last name may be formed even though the records are not physically ordered in any specific sequence. The list is referred to as a "key-sequenced" list in cases where, as here, the sequencing key (or sort key) is data stored in the boxes e.g., **231**, **241**, **251**, etc., of a table column.

The title threading pointers (**234**, **244**, **254**) of each record may be used to form a different key-sequenced path in which books are examined according to subject matter or alphabetically according to the book's title or according to some other ordering algorithm. The location threading pointers (**236**, **246**, **256**) can be similarly used to create a key-sequenced list which will identify what book is physically located next to what other book on the library's shelves.

For the sake of illustrative simplicity, only one threading pointer (i.e., **232**) is shown attached to each real data item (i.e. **231**) of each record, but it should be apparent that the author's name **231** may have many threading pointers, one for threading alphabetically according to last name, and others for threading according to additional relations such as geographic location, age, number of published books and so forth. It is up to the computer programmer and the access control program **120***d* to assign "meaning" to each box and thus determine whether that box will function as a storage area for real data or for ancillary data such as pointer data.

The records of FIG. **2**A may be visualized as being serially stacked one on the next according to a sequence defined by a preselected one of the threading pointers (e.g. **232** or **234** or **236**) to thereby create a displayable table which has as entries in the columns of each row, the real data items: author's name **231**, book's title **233** and book's location **235**. The ancillary threading pointers **232**, **234**, **236** are hidden from the lay user's view. New rows are added to the table by breaking a logical link (e.g., $L_{11}$) between a preceding pointer (e.g. **232**) and a next pointer (e.g. **252**) to insert a new record in the search path. The rows can be of variable length since the linking address pointers can point to any arbitrary location in the bulk memory means **130**. To get to the $N^{th}$ item of a threaded list, one normally sequences from the beginning of the list (table) through all the threading pointers until the $N^{th}$ access is performed, at which point the contents of the addressed record area can then be read. For relatively large tables (e.g. those having thousands of rows), this process of sequencing through all the threading pointers to reach the $N^{th}$ row of a table can take a significant amount of time.

Referring to a second embodiment **260** shown in FIG. **2**B, the structure of an older and less sophisticated data organizing system will be described. In a bulk memory section **130***b* of this older system **260**, data is organized according to what is commonly referred to as "relative table" addressing. Threading pointers are not used for logically linking one record (row) to the next. Instead, each data string (e.g., **270**) can be shrunk to contain only the essential target information, such as in this example, author's name (**271**), book's title (**273**) and book's location (**275**), with one item of real data being physically located adjacent to the next. The examination of all record items in this structure **260** may be performed according to the physical location of each record (**270**) within the address space of bulk storage area **130***b* (the next adjacent string **280** follows first string **270** and so forth). Unlike the purely key-sequenced organization of FIG. **2**A, the physical proximity links $PR_{012}$, $PR_{023}$, $PR_{034}$, etc., of FIG. **2**B do indicate a particular ordering of the stored information.

The relative-table organization is somewhat similar to the way that index cards are physically ordered in a manual library system according to author's last name, except that the library catalog trays should now be visualized as having sequentially arranged grooves defined on their bottom-inner surfaces. Each groove is numbered according to its absolute position and only one card can be slotted into each groove. With this system, each card can be immediately located by its groove number rather than by thumbing through the information of all previous cards. If a groove number is known, substantial time can be saved in locating the corresponding card and obtaining the information written on its face. If the groove number is not known, the same relative-table organization can be searched by sequentially thumbing through the trays and examining the cards according to a key-sequenced approach in order to find a desired card even though the cards are stored in grooves. The relative-table organizing method is not mutually exclusive of a key-sequenced examination method. There is a difference between a purely key-sequenced table and a relative table, however. A relative-table organized system is not as easily updated as is a purely key-sequenced system. In the relative table system, a new card cannot be inserted between two cards which already fill adjacent slots. This inflexibility has led many in the database management field away from the relative-table method and towards purely key-sequenced systems since the latter can accept any number of new cards for insertion between old cards.

In FIG. **2**B, all the record areas are of a fixed and predefined length. The fixed length of each record defines the groove size. To access the $N^{th}$ item of a "relative-table" type of list **130***b*, one need only multiply the fixed record length by the value N to directly obtain the physical address (slot) of the desired record. There is no need to sequence through a chain of threading pointers in order to find a desired row once its slot number (groove number) is known. Empty slots **290**, such as the slot number 4 shown in FIG. **2**B, are preferably scattered throughout the address space of the bulk memory section **130***b* to allow for occasional insertion of new items.

It should be noted that while the relative table organization **130***b* of FIG. **2**B is neither as flexible nor as easily updated as the key-sequenced organization **130***a* of FIG. **2**A, the relative-table structure **130***b* has one major advantage over the key-sequenced structure **130***a*; an $N^{th}$ item in a relative-table list **130***b* may be accessed much faster than the $N^{th}$ item of a key-sequenced list **130***a*.